

NAVAL POSTGRADUATE SCHOOL MONTEREY, CALIFORNIA



THESIS



**ON DESIGN AND ANALYSIS OF A
DIFFERENTIAL GLOBAL POSITIONING
SYSTEM (DGPS) AIDED NAVIGATION SYSTEM
FOR AN UNMANNED AIRBORNE VEHICLE**

by

Alexandros Christofis

June, 1995

Thesis Advisor:

Isaac Kaminer

Approved for public release; distribution is unlimited.

19960117 016

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704	
<p>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.</p>				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE June 1995		3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE ON DESIGN AND ANALYSIS OF A DIFFERENTIAL GLOBAL POSITIONING SYSTEM (DGPS) AIDED NAVIGATION SYSTEM FOR AN UNMANNED AIRBORNE VEHICLE				5. FUNDING NUMBERS
6. AUTHOR Christofis Alexandros				
7. PERFORMING ORGANIZATION NAME(S) ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000				8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited				12b. DISTRIBUTION CODE
13. ABSTRACT <i>The purpose of this thesis is to investigate using Global Positioning System (GPS) technology, for the guidance of an unmanned air vehicle (UAV) seeking precise navigation. Being fully operationally deployed by now, GPS is the indispensable tool for every application that requires precise positioning. By applying the Differential Positioning Technique, GPS becomes accurate to the point that its position and velocity outputs can be used as inputs to a flight control computer of a UAV. The Archytas unmanned airborne vehicle at the Naval Postgraduate School will achieve autonomous flight using a Texas Instruments TMS320C30 Digital Signal Processor. The latter is hosted on an IBM compatible PC, and is controlled via Integrated System's AC100 control system design and implementation software package. The GPS receiver used throughout this thesis is a Motorola PVT-6 OEM. Another identical receiver is used as a reference station, enhancing the former with Differential capability. Finally, the achieved accuracy of the differential setup is examined with respect to the Archytas flight controller requirements.</i>				
14. SUBJECT TERMS Differential Global Positioning System, Navigation, Unmanned Airborne Vehicle, Autonomous Flight, Flight Controller.				15. NUMBER OF PAGES 112
				16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

Approved for public release; distribution is unlimited.

**ON DESIGN AND ANALYSIS OF A DIFFERENTIAL GLOBAL POSITIONING
SYSTEM (DGPS) AIDED NAVIGATION SYSTEM FOR AN UNMANNED
AIRBORNE VEHICLE**

Alexandros Christofis
Lieutenant Junior Grade, Hellenic Navy
B.S., Hellenic Naval Academy, 1987

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN AERONAUTICAL ENGINEERING

from the

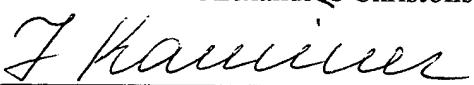
NAVAL POSTGRADUATE SCHOOL


June 1995

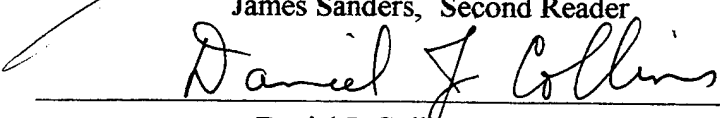
Author:


Alexandros Christofis

Approved by:


Isaac I. Kaminer, Thesis Advisor


James Sanders, Second Reader


Daniel J. Collins, Chairman,
Department of Aeronautics and Astronautics

ABSTRACT

The purpose of this thesis is to investigate using Global Positioning System (GPS) technology, for the guidance of an unmanned air vehicle (UAV) seeking precise navigation. Being fully operationally deployed by now, GPS is the indispensable tool for every application that requires precise positioning. By applying the Differential Positioning technique, GPS becomes accurate to the point that its position and velocity outputs can be used as inputs to a flight control computer of a UAV.

The Archytas unmanned airborne vehicle at the Naval Postgraduate School will achieve autonomous flight using a Texas Instruments TMS320C30 Digital Signal Processor. The latter is hosted on an IBM compatible PC, and is controlled via Integrated System's AC100 control system design and implementation software package.

The GPS receiver used throughout this thesis is a Motorola PVT-6 OEM. Another identical GPS receiver is used as a reference station, enhancing the former with Differential capability. Finally, the achieved accuracy of the differential setup is examined with respect to the Archytas flight controller requirements.

Accession For	
NTIS CRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

TABLE OF CONTENTS

I. INTRODUCTION	1
II. THE GLOBAL POSITIONING SYSTEM	5
A. THE SPACE SEGMENT	5
B. THE CONTROL SEGMENT	6
C. THE USER SEGMENT	7
III. USING THE GPS POSITIONING INFORMATION	11
A. USING THE POSITION FIX OUTPUTS	11
B. USING THE RAW GPS OBSERVABLES OUTPUT	15
IV. HARDWARE DESCRIPTION	19
A. MOTOROLA PVT-6 DESCRIPTION	19
B. ARCHYTA'S FLIGHT CONTROLLER	21
V. INTERFACING THE GPS RECEIVER WITH THE AC100 MODEL C30 FLIGHT CONTROLLER	23
A. AC100 MODEL DESCRIPTION AND FUNCTIONALITY	23
B. PVT-6 GPS RECEIVER OUTPUT MESSAGE	24
C. THE "USER_SER.C" USER INTERFACE FILE	27
VI. SETTING UP THE DGPS	31
A. GPS RECEIVERS INITIALIZATION AND CONFIGURATION	31
B. HARDWARE CONNECTIONS	35
VII. ON THE FIELD TESTING AND PERFORMANCE EVALUATION	37
A. EQUIPMENT SETUP	37
B. NON DIFFERENTIAL POSITIONING DATA	37
C. DIFFERENTIAL POSITIONING DATA	38
VIII. CONCLUSION	41
APPENDIX A. FIGURES	43
APPENDIX B. TABLES	77

APPENDIX C. USER_SER.C-GPS PROJECT	80
APPENDIX D. USER_SER.C-BK PROJECT	96
LIST OF REFERENCES	101
INITIAL DISTRIBUTION LIST	103

I. INTRODUCTION

Since the dawn of civilization, navigation was one of the big concerns of human beings. The question "where am I?" or "where am I going?", was vital for survival whenever primitive man left his settlement to fulfill his everyday needs or esoteric motivations.

During his long struggle to explore and conquer the globe, man has many times turned his eyes to the sky looking for help with his navigation. The stars could help a navigator find his way from one continent to another but could not help him avoid deadly reefs, or find a harbor entrance in the night. Many navigation tools have been used since then, some more efficient than others, but today man once more turns his eyes to the sky in order to solve once and for all his navigation problems around the globe. He is now aided by man made stars.

A true revolution in the world of navigation, the Global Positioning System (GPS) was born in 1973 after a small group of Armed Forces officers and civilians completed a plan which proposed that precise navigation could be achieved by using radio-ranging measurements to a constellation of artificial satellites called NAVSTAR's.

Two decades later, that early plan became a reality. Twenty four on orbit satellites give GPS its full operational capability, providing uninterrupted signal coverage around the globe and changing many of the traditional ways that man obtained his positioning. The system is owned by the United States Air Force and managed jointly by the Department of Defense and the Department of Transportation. As soon as the system became available to the public, it was embraced by all those interested in precise navigation, becoming a unique utility which is, day by day, pushing aside the existing navigation systems.

The aviation community is one of the Global Positioning System's most dedicated users. The military is taking advantage of the system's full operational capabilities which allow it to pinpoint aerial vehicle's position with an accuracy of about seven meters, while the civilian users are offered a navigation signal of degraded accuracy, with an average error of about one hundred meters. The civilian accessible signal is transmitted at a different frequency than the military one, with its accuracy intentionally degraded by introducing errors in several critical navigational parameters transmitted by it. This is in response to the Department of Defense's concerns for preventing unauthorized use of this navigation utility by hostile forces against the United States.

To compensate for this degradation, a corrective technique has been introduced which is called Differential GPS. The idea is that a stationary receiver, (called base station), with its position known, can extract the needed correction for the erroneous navigational parameters transmitted by the GPS signal, by comparing the position fix that it calculates from the satellite signal with its *a priori* known position. These corrections are good for receivers located within several hundred miles from the base station and can be made available by radio broadcast.

The accuracy of determining position using the Differential GPS (DGPS) technique has a typical value of less than three meters. With such a level of accuracy, DGPS is looking particularly elegant in applications where precise positioning information is important, such as the flight control of an unmanned aerial vehicle (UAV). To ensure continuous and smooth control of the UAV's flight path, the GPS provided position fix should be blended with the outputs of an inertial measurement unit. This is mainly because of the following reasons:

- The currently manufactured GPS receiver units output position information usually once per second, a rate that is insufficient to provide real time position of the vehicle when the latter experiences large accelerations.
- Due to temporary signal loss that can happen for various reasons, the GPS receiver may not be able to provide navigation solution for a period of time. Although the time length associated with such situations is usually not more than a few seconds, is still enough to cause the loss of the vehicle's position control.
- The positioning solution offered by GPS is "wandering around" due to various navigational errors, thus creating the necessity for the integration with the more stable inertial measurement unit.

The research effort described in this document is to provide three dimensional positioning information to the computer controlling the flight of the Archytas UAV at the Naval Postgraduate School, using Differential Global Positioning System technology. In the following chapters, we discuss the GPS receivers used, the flight control computer, and the way in which they are interconnected. The appendices include the receiver specifications, the figures, as well the C code needed to interface between the two machines i.e., the GPS receiver and the flight control computer.

II. THE GLOBAL POSITIONING SYSTEM

In order to fully understand and evaluate the positioning information obtained by the GPS, its principles of operation will be discussed. The system consists of three major segments:

- The space segment
- The control segment
- The user segment

Although the last segment is apparently the most interesting for the present application, the other two segments are also presented. The whole system is presented in Fig.2.1.

A. THE SPACE SEGMENT

The fully operational space segment consists of twenty four satellites that are placed in six different orbital planes at an altitude of 10,898 nautical miles above the surface of the earth. The first satellite, model Block I, was launched on February 22, 1978 and is no longer operational. Between February 1978 and October 1985 there were ten more identical satellites launched among which only one is still operational. From February 1989 to October 1990 nine more vehicles were put on orbit designated as type Block II. From November 1990 to March 1994 the last generation of fifteen GPS satellites designated as type Block IIA were launched into orbit. The twenty four Block II and Block IIA vehicles are those which constitute the full operational GPS constellation. Three of them are on-orbit spares (Fig.2.2).

The information provided by the satellites to the users is sent to earth by the means of two L-band carrier signals, L1 (1575.42 MHz), and L2 (1227.6 MHz). On these carrier frequencies are superimposed two different binary codes, the C/A code (which stands for Coarse Acquisition), and the

P-code (which stands for Precision). The P-code is only accessible by military users and is protected by encryption so that full positioning accuracy is denied to unauthorized users. Although all satellites are transmitting on the same frequency, they are assigned their own unique C/A and P codes so that the GPS receivers can distinguish between them.

In an ideal world, the satellites should stay right on their orbits and continue to rotate around the earth in accordance with the Keplerian laws. Unfortunately this is not the case since their orbits are disturbed by various forces. The most important ones are the gravitational perturbation of the earth's second and fourth zonal harmonic, solar and lunar gravity, solar radiation pressure, and various gravity anomalies. Therefore, there exists the need for a control center that closely follows each satellite's behavior and take the required corrective actions when necessary. This is exactly the task of the control segment.

B. THE CONTROL SEGMENT

The control segment has the sole responsibility to make sure that the GPS satellites are in their proper orbit , functioning correctly, and transmitting the correct values of the navigational parameters. The GPS ground network consists of five active-tracking ground antennas and five passive-tracking monitor stations, located around the world.

The active-tracking ground antennas actively track the GPS satellites, transmitting commands and navigation uploads, and recording telemetry over S-band links. The passive tracking monitor stations passively track the L-band signals transmitted by the satellites to determin the vehicle's navigational data.

All the above data is collected at the Falcon Air Force Base, Colorado Springs, Colorado, where the master control station of the GPS control segment is located. The collected data is used as an input to a Kalman Filter

from which the orbital states of the space vehicles can be determined. New navigational parameters are then passed to the satellites via the S-band radio links. The master control station personnel are also responsible for maneuvering the satellites to keep them in their preassigned orbits. When satellite's performance is inaccurate or a mechanical failure has occurred, the vehicle is considered "unhealthy" and a special warning is incorporated in its navigational message to warn the GPS users.

With the whole GPS infrastructure perfectly working, the only component that remains in order for a navigator to obtain a position fix is a GPS receiver which will receive, decode, and process the navigational data transmitted by the satellites. The user's end of the GPS, the GPS receivers, is referred to as the user segment.

C. THE USER SEGMENT

The user segment is responsible for obtaining the navigational signals provided by the satellites and extracting the precise values of three dimensional position, velocity, and time. There are several types of receivers. Their classification is usually by how many channels they have.

Each separate channel is a radio receiver that can track one or more satellites. If it tracks one satellite, it is full time dedicated to that one signal. If it tracks more than one satellite, it has to use time sharing technique, that is, tracking each satellite sequentially for a specified amount of time. The time sharing technique yields poorer results and it was used by the very first receivers built. A typical number of separate channels for a modern receiver is six to twelve.

There are two major observables that can be used by a receiver in order to determine its position. The first is the pseudorange and the second is the carrier phase. Each satellite time-tags its transmissions so that when they are

received by the receiver, the latter knows the transmission time. The receiver also knows the time that the transmission was received from its internal clock. The receiver software then extracts the time difference between transmission and reception, that is the time it takes the signal to travel from satellite to the receiver. Multiplying this time difference by the speed of light, the actual distance between the satellite and the receiver can be calculated.

Onboard the satellites, extremely precise and expensive atomic clocks keep track of GPS time. On the other end, the receivers carry relatively cheap clocks which are not very accurate. A one billionth of a second uncertainty in the time difference measurement multiplied by the speed of light yields a range uncertainty of one foot. So the receiver clock inaccuracy is an error that has to be taken care of during the range determination. The calculated range is called pseudorange since it is not the actual range due to the above error.

By decoding the satellite message, the GPS receiver can also obtain the satellite's position (ephemeris) in earth-centered earth-fixed coordinates. By doing this for four satellites it can solve the following set of equations and determine its position coordinates x , y and z , along with the receiver clock error. All position coordinates x, y, z are given in earth-centered earth-fixed coordinate system:

$$R_1 = \sqrt{(x_1 - x)^2 + (y_1 - y)^2 + (z_1 - z)^2} + c * dt$$

$$R_2 = \sqrt{(x_2 - x)^2 + (y_2 - y)^2 + (z_2 - z)^2} + c * dt$$

$$R_3 = \sqrt{(x_3 - x)^2 + (y_3 - y)^2 + (z_3 - z)^2} + c * dt$$

$$R_4 = \sqrt{(x_4 - x)^2 + (y_4 - y)^2 + (z_4 - z)^2} + c * dt \quad (2.1)$$

where R_i is the pseudoranges, dt is the receiver clock error and c is the speed of light. The subscripts denote quantities related with each of the four satellites.

Next, the position fix of the receiver obtained in earth-centered earth-fixed coordinates must be transformed to the user's preferable coordinate system. It should be noted that the above equations (2.1) represent a very basic problem formulation. In real life, the receiver's software has to take care of many other factors that are involved in the problem solution. In fact many of the problem parameters are obtained as states of a Kalman filter, rather than straightforward algebraic solutions.

Use of the carrier phase observable is a relatively new technique. When a GPS receiver acquires the signal of a satellite, it can measure the fractional part of a single cycle of the carrier wave. As every complete cycle represents distance equal to the wavelength of the carrier wave, (20 cm for the L1 frequency), the carrier phase observable can be translated to range information from the particular satellite. Of course we cannot determine a pseudorange only from carrier phase measurements because it is impossible to know how many complete cycles are there between the satellite and the receiver without any other information. What we can do is track the carrier phase changes, and feed them into the Kalman filter mentioned above to "smooth" pseudorange estimates.

To determine its velocity, the receiver also monitors the carrier wave. This time, the Doppler shift frequency from each tracked satellite's carrier wave is monitored. The relative velocities with respect to the satellites are determined, and the receiver's velocity, resolved on the three dimensional coordinate system, is computed from the known satellite positions. The GPS

receivers do not calculate velocity by working out a time-distance problem between two consecutive position fixes as a common misconception suggests.

III. USING THE GPS POSITIONING INFORMATION

There are two approaches with respect to what form of GPS information can we use to obtain precise positioning. The first, and most trivial, is to use the readily available position fix coordinates (in whatever coordinate system) that the receiver is providing after processing the raw GPS observables, i.e. the pseudoranges and the carrier phase. In the second approach, the user develops his own software that processes the raw observables which are available as outputs of many GPS receivers to obtain his application specific information.

In our case, we are using two OEM (original equipment for manufacturers) class receiver modules type PVT-6 made by Motorola. One of them will be placed on the UAV while the other one will be stationed on the ground and used as a reference station providing differential corrections for the operational area.

Both ways are evaluated here with respect to our specific Archytas application.

A. USING THE POSITION FIX OUTPUTS

The Motorola PVT-6 receivers use three coordinate systems in order to determine their position fix and their velocity. These are:

1. Geodetic Latitude, Longitude, Height.

The first is the well known Geodetic Latitude, Longitude and Height. The geodetic latitude of a point is the angle from the equatorial plane to the vertical direction of a line normal to the reference ellipsoid. The geodetic longitude of a point is the angle between the Prime Meridian plane and the plane passing through the point, both planes being perpendicular to the

equatorial plane. The geodetic height of a point is the vertical distance, (with respect to the reference ellipsoid), from the point to the reference ellipsoid itself.

The above three coordinates can have different values depending on the reference ellipsoid and geodetic datum that is used to define the coordinate system. The reference ellipsoids are ellipsoidal models of the shape of the earth. They are defined by their semi-major axis, (equatorial radius), and their inverse flattening parameter. The most accurate of them is the WGS-84 which models the shape of the earth over a smooth averaged sea surface with an accuracy of about one hundred meters [Ref.9].

The reference systems describing the size and shape of the earth are defined by the geodetic datums. The geodetic datums obtained from a reference ellipsoid and the relationship between the ellipsoid and a point on the topographic surface established as the origin of the datum. This relationship can be defined by six quantities generally, (but not necessarily), which are the geodetic latitude, longitude and the height of the origin, the two components of the deflection of the vertical at the origin, and the geodetic azimuth of a line from the origin to some other point [Ref.9].

There are many different datums used around the world so that one point on the surface of the earth can be described by different set of values for latitude, longitude and height, depending in which datum are the coordinates expressed. Relating geodetic coordinates with the wrong datum, can result in a position error of hundreds of meters. The most recent and accurate of all the datums is the WGS-84 (World Geodetic System - 1984), which is used as a common ground in order to unify all the different datums used around the world. The WGS-84 has been used by the GPS since 1987.

Finally, we should note that the receiver outputs its velocity as a vector with magnitude expressed in meters per second and direction expressed in degrees with respect to the true (WGS-84) north.

2. Earth Centered Earth Fixed Coordinate System (ECEF)

This coordinate system is used by our Motorola GPS receivers in order to alternatively express three dimensional position. This is a Cartesian coordinate system with respect to the center of mass of the reference ellipsoid where:

- The Z-axis points towards the North Pole.
- The X-axis is defined by the intersection of the plane defined by the prime meridian and the equatorial plane .
- The Y-axis completes a right handed orthogonal system by a plane 90 east of the X-axis and its intersection with the equator.

Again here the reference ellipsoid is the WGS-84. The velocity is not expressed in the ECEF coordinate system, however it is expressed in the tangent plane coordinate system.

3. Tangent Plane (Local Geodetic) Cartesian Coordinate System.

This coordinate system can be defined at any point of the surface of the earth by considering a plane tangent to the reference ellipsoid at that point and:

- The X-axis being on the plane and pointing towards true East.
- The Y-axis being on the same plane and pointing towards true North.
- The Z-axis being perpendicular to the tangent plane and pointing towards the center of the ellipsoid.
- All the axis originate from the common point of the ellipsoid and the tangent plane.

The tangent plane system is used by the Motorola receivers in order to provide an alternate expression for the velocity. The velocity is expressed by means of North, East, and Up components. It should be noted that the Up

component is positive when it is directed away from the center of the reference ellipsoid, i.e. along the usual upward direction.

4. Pros And Cons Of Using The Position Fix Outputs

The easiest way to obtain position and velocity data for Archytas is to use the readily available position and velocity information in the format discussed above. It also should be noted that, since the velocity is expressed in the local tangent coordinate system, the integration between the GPS and the inertial measurement unit (IMU) data is very easy since the latter also uses the same coordinate system not just to express velocity, but also acceleration and position. The IMU data is used by the Archyta's flight controller as the primary information to solve the bird's stability problem.

To take advantage of the Differential Correction technique, we have to use two GPS receivers simultaneously. The first one will be placed on Archytas and will transmit position and velocity to the flight control computer on the ground by a radio link. The second GPS receiver will be on the ground functioning as a reference station that will generate differential corrections for the GPS raw observables (pseudorange and pseudorange rates). These corrections will then be uploaded in real time to the vehicle's GPS receiver by means of another radio link, thus providing the full differential capability.

There are two choices for setting up the radio link system. The need of establishing a two way communication system is obvious. This can be done either in full or half duplex.

For the half duplex mode, we need two radio link transceivers, one on the vehicle and another on the ground. Both transceivers will transmit on the same frequency. Next we need two computers, again one on the vehicle and one on the ground. These computers will play the role of the "traffic cops" for the two transceivers. Each computer will control its assigned transceiver and

will exchange the proper messages with the other computer via the radio link to ensure that when one transceiver is transmitting the other is ready to receive, and also that the two transceivers are not transmitting on the same time, which would result in loss of data.

This setup is has a major drawback. The transceiver controlled by computer onboard the vehicle would result in extra weight and extra power consumption. Therefore, the half duplex method was abandoned and the full duplex method examined instead.

The full duplex mode is straightforward to implement. The only hardware needed is two full duplex capable transceivers that will be directly communicating with the RS-232 port of the GPS receivers. Two separate frequencies will support the two way communication so that the data flow will be in real time and uninterrupted, both from the Archyta's GPS receiver to the flight controller, and from the reference station GPS receiver to the Archyta's GPS receiver.

The only drawback of this method is the cost associated with the purchase of the full duplex capable transceivers. However, these were not available during this thesis experimental work. To overcome this problem, the second more elaborated approach of using the GPS raw observables was examined.

B. USING THE RAW GPS OBSERVABLES OUTPUT

With this approach, the raw GPS observables (pseudoranges only in our case) were used in order to obtain position fix information for our vehicle. By computing the pseudoranges from at least four satellites tracked by the GPS receiver and by knowing the satellites exact position in ECEF coordinates, the position equations (2.1) can be solved yielding the receiver's position in ECEF coordinates. The parameters required to solve (2.1) are examined next.

1. The Ephemeris Satellite Message

As discussed in Chapter II, many environmental factors, some predictable and other not, tend to "derail" the satellites from their commanded orbits. For the GPS receivers to solve the position equations (2.1), the satellites positions are continuously transmitted as part of the GPS navigational message. Each space vehicle transmits its own ephemeris, that is the part of the message that describes its position, consisting of sixteen constants. These sixteen constants are:

- M_o , Mean anomaly
- n , Mean motion difference
- \sqrt{a} , Square root of semi-major axis
- Ω_o , Right ascension
- i_o , Inclination
- ω , Argument of perigee
- $\frac{d}{dt}\Omega$, Rate of right ascension
- $\frac{d}{dt}i$, Rate of inclination
- C_{uc}, C_{us} , Correction terms to arg. of latitude
- C_{rc}, C_{rs} , Correction terms to orbital radius
- C_{ic}, C_{is} , Correction terms to inclination
- t_o , Ephemeris reference time

These constants are contained in the subframes two and three of the 50-bit per second bitstream superimposed on the carrier frequency of the GPS navigation message. They are estimated and uploaded daily by the master control station at the Falcon Air Force Base for each GPS satellite, and then are retransmitted to the users of the system worldwide.

The exact position of the above constants in the satellite data bitstream is described by the **Interface Control Document ICD-200** which is published by the GPS Joint Program Office, Los Angeles Air Force Base, Los Angeles.

To compute the satellites position in ECEF coordinates from the ephemeris constants, a simple set of algebraic and trigonometric equations has to be solved.

It should be noted here that the ephemeris message is not the only GPS message containing information about satellite positions. The almanac message also contains positions, although not as accurate as in the ephemeris message. Each satellite in its almanac, transmits all the other satellite's position. However these positions are not used for position fix estimation. When the receiver acquires one satellite, then it uses the almanac information to acquire the rest that are visible in its area.

2. The Satellite Pseudoranges

The satellite pseudoranges are computed by multiplying the speed of light by the time that the signal traveled from the satellite to the GPS receiver. The travel time is found by subtracting the time that the signal was received, from the time it was transmitted from the satellite. The latter time is provided by the GPS message, while the former is obtained from the receiver's internal clock.

3. Evaluation Of Using Raw Observables Processing Approach

By now we have determined all the required parameters in order to solve the equations (2.1) and extract the receiver's ECEF coordinates. The solution can be worked out by a user developed software executed on the flight control computer of the Archytas, which is located on the ground and communicates with the vehicle via a radio link.

Why would anyone use this elaborate approach to determine the vehicle's position when the latter is readily available as a GPS receiver output? The answer is because this way a simpler radio link in simplex mode would be required, resulting in weight and cost savings. The GPS receiver onboard the vehicle would be sending to the flight controller the satellite signal

transmission time, the time that this signal was received, and the ephemeris data. The flight controller would compute the pseudoranges and the satellite positions as described above. The pseudorange corrections would be fed to the flight controller by the other GPS receiver stationed on the ground and used as a reference station by means of wired connections. Then the pseudoranges would be corrected and equations (2.1) could be solved yielding differentially corrected positions.

Although this approach sounded promising, it was abandoned for the following reasons:

- The ephemeris message is very complicated and hard to decode.
- Although satellite pseudoranges can be algebraically computed, the solutions incorporate a large number of errors. Some of them are associated with time delays that are difficult to model, and others with the presence of relativistic effects during the transmission of the signal by the satellite. The actual receiver software "smooth" the pseudoranges by using Kalman filtering technique and the carrier phase observables.
- The receiver's velocity cannot be determined because the Doppler frequency shift associated with each satellite is not available as a GPS receiver output.

As a result of all the above considerations, it was decided that the flight controller of the vehicle would be given readily available position fix and velocity information from the GPS receiver, while the differential setup would be implemented using a full duplex radio link as discussed earlier in this chapter. Because a full duplex radio link was not available during the experimental work, wired connections have been used to test the whole setup, assuming that the wireless communication is a task easily achievable using the proper commercial equipment.

Finally the software that computes the satellite pseudoranges and pseudorange rates was also developed in order to facilitate future work on the subject.

IV. HARDWARE DESCRIPTION

A. MOTOROLA PVT-6 RECEIVER DESCRIPTION

The Motorola PVT-6 GPS receiver belongs to the large class of receivers designated as OEM receiver modules. OEM stands for Original Equipment for Manufacturers and means that this receiver is not readily usable by itself, but designed to be used as a part of a larger group of devices. The first thing to be noted is that there is no display screen or keyboard. This receiver is configured, controlled and read through another computer which acts as the host equipment and communicates with the PVT-6 through a standard RS-232C interface.

Its generalized outputs are position, velocity, time, and satellite tracking status. It is capable of supporting land, air, and marine applications. Generally the requirements that distinguish each of the above applications are the Time to First Fix (TTFF), the reacquisition time, and the dynamic positioning requirements.

The TTFF defines the time before the first fix after the receiver is powered up. Reacquisition time is the time that the receiver needs in order to recompute its position, after the signal from one or more satellites tracked is lost. Dynamic Positioning is the ability of the receiver to keep track of the vehicle's position while experiencing large accelerations or decelerations.

The above requirements become more stringent for marine and air applications. The PVT-6 receiver, partly due to the six parallel channels (each one dedicated to track only one satellite) that it has, can qualify for all the above types of applications.

There are five factory options for specialized applications. These are designated as A,B,C,D,E. Our receivers are option ABC which gives us full operational capability, such as one Pulse Per Second output (1PPS), Real Time

Differential (RTD) GPS capability, and satellite Pseudorange/Carrier Phase data.

The internal components of the PVT-6 receiver are shown in Fig.4.1. The module receives the L1 frequency GPS signal, thus obtaining the C/A code (Coarse/Aquisition). The code tracking is carrier aided. A low profile, micro-strip patch antenna is used to track the L1 frequency. The antenna is shown in Fig.4.2, together with its gain pattern. It is obvious that the gain is maximum directly overhead while is diminishing at low elevations. This is in accordance with the nature of the GPS signal, which acquires large propagation errors through the atmosphere when a satellite "sees" a receiver through a low elevation angle. In the antenna module are a narrow band pass filter and a preamplifier. The preamplifier is powered with +5V DC through the coaxial interconnecting cable, which is primarily used for the signal transmission to the receiver module. [Ref.2]

Before the receiver module, the signal enters the RF Down Converter which converts it into an intermediate frequency (IF) signal. It then passes through the 6-channel code and carrier correlator section where the IF signal is converted to a digital sequence prior to channel separation. The conversion is executed by a single, high speed analog to digital converter. The resulting digitized IF signal is routed to the digital signal processor where it is split into six separate channels for code correlation, filtering, code tracking, and signal detection. [Ref.2]

The next station is the position microprocessor. This is the "brain" of the receiver as it controls its operating modes and processes all the satellite data in order to compute position, velocity, and time. Here also is the required interface to the RS232 serial port used for full-duplex, asynchronous data communication with the host equipment. [Ref.2]

The PVT-6 is characterized as Data Communications Equipment (DCE), while the host equipment is characterized as Data Terminal Equipment (DTE), in accordance with the RS-232 communications standard.

The PVT's serial port is shown in Fig.4.3. Out of the ten available pins, the only ones dealing with RS-232 type signals are pins eight, nine and ten. These are designated as transmit(TXD), receive (RXD), and ground or return (RTN) respectively. The host equipment has to have a serial channel completely devoted in monitoring the PVT-6, as the software or hardware flow control between the two devices is obviously not possible.

The PVT-6 can obtain uninterrupted navigational solutions when its velocity is less than 515 m/s and is not experiencing accelerations more than 2g. If the acceleration is less than 4g it will maintain lock on satellites. All these limits are more than adequate for our application.

Finally we note that the module is powered either with a 5V DC regulated voltage or with a 12V DC unregulated voltage. The power consumption in any case does not exceed 1.8 watts. In Figs.4.4 and 4.5, two other views of the receiver are shown. In Table 4.1, the general product specifications are listed.

B. ARCHYTA'S FLIGHT CONTROLLER

The processor used to control Archytas is a TMS320C30 type digital signal processor manufactured by Texas Instruments. It has a 40-ns cycle time enabling it to perform 60 million floating point operations per second (MFLOPS), and 30 million instructions per second (MIPS), a performance previously typical of a supercomputer. [Ref.4]

The C30 processor board is hosted on a IBM PC/AT computer, the AC100 Model C30. The basic AC100 Model C30 PC hardware consists of the TMS320C30 CPU card, a DSPFLEX carrier board, and an ethernet adapter.

The DSPFLEX board, (Fig.4.6), is a specialized full length ISA real time input-output card which functions as the interface for the C30 to communicate with four GreenSpring Computers IndustyPack (IP), or IP compatible, input-output modules. The IP modules are the interface of the controller with the "outside world". Up to four DSPFLEX boards can be installed. Each board labels its four positions for IP modules as A,B,C,D (Fig.4.7). The IP modules are accordingly labeled 1,2,3,4 corresponding to the above positions. This is particularly important, since the modules are referred to by the software using these numbers. [Ref.5]

Many types of IP modules are available. A serial one is used to interface the GPS receiver with the C30 and is assigned the module number 3, (position C on the DSPFLEX board). It provides two channels of multimode serial communication in both the RS-232C and RS-422 standards. There are two physical serial channels on each IP serial module, named channelA and channelB, communicating via a 50-pin connector. In our case, that means the C30 can "read" two GPS receivers at the same time. The pin-out configuration of the IP serial module is shown in Fig.4.8. [Ref.5]

The complete C30 system also includes a UNIX workstation. The workstation is used to run specific software packages such as Integrated System's SystemBuild design and analysis software, and AutoCode automatic code generator. These allow for the various algorithms to be developed in block diagram form, then transformed into C code, and executed in real time on the C30 DSP board. During real time operation, the UNIX workstation can be used to acquire data and graphically interact with the executing application. The complete hardware setup is shown in Fig.4.9. [Ref.5]

V. INTERFACING THE GPS RECEIVER WITH THE AC100 MODEL C30 FLIGHT CONTROLLER

A.AC100 MODEL DESCRIPTION AND FUNCTIONALITY

To interface the GPS receiver with the AC100 Model C30 controller, a simple block diagram description of the system had to be built using the AC100 software package installed on the UNIX workstation. Then this model was converted into an executable C code file and downloaded on the C30 for execution. Any manipulation of the GPS receiver data is unnecessary at this point since we are only interested in simply reading it and verify its correctness. Thus, a unity gain block was developed in SystemBuild (Fig.5.1).

The general idea is that the GPS data is transmitted from the GPS receiver's RS-232C serial port (by wired connections, or wireless by radio link) and is received by the IP-Serial module (channelA or channelB) of the C30 controller. Then it is processed as specified in the file "user_ser.c", which is the user interface to the IP-Serial Device Driver. After data is decoded, it is displayed on the UNIX workstation using a graphical user interface (GUI).

The GUI is built by invoking the Interactive Animation Builder. Four icons are built in order to have a better representation of the GPS data. Each icon displays the variables originating from a specific GPS receiver message type. The first three icons show the data from the GPS receiver which is supposed to be placed on Archytas, while the fourth shows data from the differential base station.

The first icon, (Fig.5.2), displays the receiver's Latitude, longitude, Height-msl (mean sea level), Height-GPS (with respect to the reference ellipsoid, in our case the WGS-84), Velocity and heading. The second icon, (Fig.5.3), displays the ECEF coordinates X,Y,Z, of the receiver plus its three

velocity components in the Local Tangent Plane coordinates, i.e. North, East, and Up. The third icon, (Fig.5.4), displays the Satellite Vehicle Identification Number (SVIN) of the satellites tracked, their tracking mode byte, their GPS message transmission time, the extracted pseudoranges and pseudorange rates of each, and finally the receiver's GPS local time. The fourth icon, (Fig.5.5) displays the SVIN tracked by the reference station, the GPS time reference, and the pseudorange, pseudorange correction and the issue of ephemeris data for each.

In each icon, two "checksum" bytes are also present. These represent the internal checksum of the GPS message and the computed one from the user_ser.c file, which are used to check the received data's integrity.

It should be noted that the only thing executed by the TMS320C30 DSP processor while the model is running in real time, is the executable image of the unity gain with seventy seven input variables provided by the IP-Serial Device Driver and outputs the same seventy seven variables.

An important part of this thesis was to develop and test the user_ser.c user interface file, because therein the actual decoding of the GPS receiver message takes place. This message is then passed in a usable format to the above mentioned AC100 model. For this reason, user_ser.c is discussed next together with a receiver binary message examination.

B. PVT-6 GPS RECEIVER OUTPUT MESSAGE

As discussed earlier the receiver data is routed to the host equipment through a RS-232C serial port. For flexibility Motorola has made available three interface protocols for the PVT-6. These are the Motorola Proprietary Binary Format, the NMEA-0813, and the LORAN emulation format. The

Motorola Proprietary Format is used throughout this thesis, because of its ease of use and its unrestricted access to all the receiver's outputs.

1. Motorola Binary Format Description

In this format, the GPS receiver messages consist of a number of binary characters. All the messages start with two "@" characters (hex40). The next two characters, (one byte each), are an upper or lower case letter denoting the type of the message, i.e. the particular structure and format of the remaining binary data. Then come the bytes containing the information carried by the message. The last three bytes are always the checksum followed by a carriage return <CR> and a line feed <LF>, denoting the end of the message. The checksum is the exclusive-or of all the message bytes after the last "@" and before the checksum itself.[Ref.2]

The above format is valid for both input and output receiver messages. In the case of the input receiver messages, the user is responsible for creating the correct bytestream taking care of the message format and the checksum. The incoming messages to the receiver are placed in a buffer which is serviced once per second and is 2048 bytes long.

Both input and output messages contain data that is interpreted as unsigned integers, signed integers, or floating point values. It is the user's responsibility to correctly parse, decode, and scale the data bytestream he receives to obtain the correct results. After the message is read, a checksum should be generated in order to check if the data have been correctly received by the host equipment. The structure of all the Motorola Binary Format messages, is contained in paragraph 4.7 of Ref.2.

The basic output messages can be selected to be one-time output (polled) or at a selected update rate (continuous). The continuous messages may have a rate of once per second to once every 255 seconds. One point that has to be considered, in the case of multiple continuous messages, is that the

receiver can output only 750 bytes per second. If the number of output bytes per second is greater than 750, then the messages that don't fit into the 750 byte stream will be rescheduled for the next output cycle. The basic output messages are shown in table 5.1.[Ref.2]

The Motorola binary format also supports differential capability. That means that the pseudorange and pseudorange rate corrections messages can be expressed in the above format so that they can be interchanged between Motorola receivers operating in the proprietary format.

The industry standard format for the exchange of differential corrections between electronic devices is the one issued by the Radio Technical Commission for Maritime Services (RTCM) under the designator RTCM SC-104. This format can also be decoded from the receiver when the latter is operating under the proprietary protocol, and is especially useful when the receiver is communicating with a receiver of different make.

A typical example is a PVT-6 onboard a seagoing vessel, receiving differential corrections from the US Coast Guard DGPS coastal stations, which use GPS reference receivers manufactured by Ashtech, Inc. Table 5.2 shows the receiver's differential capability with respect to the I/O protocol used.

2. The NMEA-0813 Format

The National Marine Electronics Association (NMEA) defines the standard type of messages to be exchanged between electronic navigation devices via the RS-232C serial interface. It has limited output messages. This format is useful only when our receiver needs to be interfaced with devices using it, i.e. an autopilot, so it will not be examined further.

3. The LORAN Emulation Format

This format is used whenever the PVT-6 needs to replace LORAN receivers in embedded positioning system applications. It is of no interest in our case.

C. THE "USER_SER.C" USER INTERFACE FILE

As previously stated, this file is the most essential part of the whole project, because it carries out the actual parsing, decoding, and verification of the GPS messages. It is included in the basic AC100 software package as a user alterable interface template to the IP-Serial Device Driver, and follows a standard modular design. There are three main functions in user_ser.c that the user must develop to suit his specific application. These three functions are:

1. get_SERIAL_parameters
2. user_SERIAL_out
3. user_sample_SERIAL_in

In the following paragraphs our specific software will be explained. The program listing is shown in the Appendix C. Detailed instructions on how to use these functions are provided in pages 133-142 of the Ref.5.

1. The "get_SERIAL_parameters" Function

This function is called once during initialization of the system. It contains the user specified parameters used for the asynchronous serial communication through the IP-Serial modules. As it can be seen from the Appendix C, the parameter values are in accordance with the PVT-6 output, i.e. parity = NONE, baud_rate = 9600, stop_bits = ONE, receive_data_size = 8, clock_multiplier = 16.

2. The "user_SERIAL_out" Function

This function is used to accommodate output data and is not used in our case.

3. The "user_sample_SERIAL_in" Function

This function is composed of two parts. One that reads channelA, and one that reads channelB. It is called two times per sampling interval, one for channelA and one for channelB. The channel called specifies one of the three calling parameters of the function, the unsigned integer `ser_channel`.

The second calling parameter is the float array, `model_float[]` which is the vector to be filled with values that will transfer to the executed model as inputs. This second parameter has length equal to the number of parameters of the model that are expected as inputs from the serial IP module. Note that the two physical channels A and B, are assigning values to the same elements of the `model_float[]`, i.e. one to twenty six. This is perfectly all right, as the two physical channels are never active on the same call of "user_sample_SERIAL_in".

The GPS receiver that will be placed on Archytas is connected to channelA and outputs three messages. The first begins with the string "@@Ba" and contains the Latitude, Longitude, Height, Velocity and Heading of the receiver. At first the routine detects the string "@@Ba" and then the function `read_serial` reads from the input buffer the expected number of bytes in the message (section 4.7 of Ref.2). The message is then parsed, and the bytes that contain a specific variable are multiplied by their hexadecimal value and summed together. Then the variables that need scaling are scaled accordingly in order to obtain the specified resolution.

Next step is to create the checksum (the exclusive-or of all the bytes received after the heading string "@@Ba") and to compare it with the one

received from the message. If both checksums agree the obtained values are placed in the appropriate elements of the `model_float[]` array. If the checksums don't agree, then the last correct value of the variables is returned, being stored in the global variable `last_float[]`.

The same procedure is used to decode the messages that contain the ECEF coordinates of the receiver plus its velocity components in the tangent plane coordinate system, ("@@Bk ... "), and the message that contains the pseudorange and pseudorange rate information, ("@@Bg ... ").

The receiver used as the DGPS reference station is hooked to channelB. It produces only the pseudorange and pseudorange rates correction message, ("@@Ce ... "). This message is decoded using the usual procedure. It has to be noted that the portion of the code that reads channelB together with the portion of the code of channelA that outputs the pseudoranges and pseudorange rates, (message "@@Bg ... "), have a rather indirect value as their outputs will not be used by the Archytas flight controller at the present. The reason that have been developed is to facilitate a future approach of the positioning problem as described in Chapter III, section B.

Especially the outputs of the "@@Ce ..." message won't even show up on the screen of the UNIX monitoring station, as the signal will be fed directly to the Archytas GPS receiver during the DGPS setup. At this case the fourth icon will show the arbitrary default values of seven, denoting absence of data.

VI. SETTING UP THE DGPS

The differential GPS setup is discussed at this chapter. This will be suitable for equipment testing and performance evaluation. With a suitable radio link, this setup can be expanded to one installed on a flying vehicle.

A. GPS RECEIVERS INITIALIZATION AND CONFIGURATION

The Motorola binary communication format gives the user the capability to fully define the receiver's operating parameters and mode of operation during initialization, as well as during the receiver's normal operation. Note, our application has one special characteristic. Since the UAV will be flying away from the controlling station, the GPS receivers should be properly preconfigured, as real time control of them will not be possible with the existing setup of the Archytas controller.

A full duplex radio link will connect the UAV receiver with the base station. The differential station will send corrections to the UAV, while the UAV will report its position to the flight control computer that stays on the ground. Any desired commands to the UAV GPS receiver will have to be transmitted through the differential corrections wireless circuit, and into the PVT-6's single I/O port. To make sure that the commands and the corrections won't interfere with each other on the sole frequency provided, some form of system controller has to be installed to handle both inputs. For details see paragraph 4.4.2 of Ref.2. Fig.6.1 shows a similar setup, where a control unit makes sure that both differential corrections from a Coast Guard Station, and GPS commands, enter a PVT-6 receiver without any data collision.

Similar setup is also achievable using our AC100 model C30 based controller, however the software needed to realize it is not available at this time. Apart from that, a carefully planned GPS receiver

initialization-configuration should be sufficient for our application, so that the real time control of the unit is not required and the added complexity is avoided. A setup without real time control of the receiver is shown in Fig.6.2.

1. Configuring The UAV Receiver

An easy way to configure the PVT-6 is to use the evaluation software kit which runs on a PC and is provided by Motorola. The only thing the user is required to have is a cable connection that will merge the power supply wires and a nine or twenty five pin standard serial connector, into a socket that will be connected to the receiver's I/O ten pin port. The standard serial connector would be hooked to any of the PC's available serial ports (usually serial port number two), and the GPS program should be run. Detailed instructions for the whole setup are provided by Ref.7. The setup diagram is shown in Fig.6.3.

Although this is not the proper way to configure the PVT-6, it works fine and is also very useful in revealing the receiver's little "secrets", that one can uncover after playing around with the program for a while. First we are going to examine the UAV receiver configuration.

The receiver is forced into the Motorola Proprietary Binary format each time the GPS program is started. Following the instructions of Ref.7, page 9, we initialize the receiver. Then we use the following commands in order to configure it:

- **mode : f**, forcing the receiver to the fix mode, instead of idle.
- **ah : d**, disabling the altitude hold feature.
- **almhold : u**, the receiver updates its internal almanac when it receives a new one.
- **aptype :air**, optimizing the receiver for airborne use.

- **datum : 49**, to make sure that the receiver calculates its coordinates on the WGS-84 datum.
- **doptype : PDOP**, to tell the receiver to use the Position Dilution of Precision type for its satellite selection criteria (instead of geometrical, horizontal, vertical or time dilution of precision). The DOP is a figure that characterizes a given combination of GPS satellites. Depending on the relative satellite position, a given combination increases the uncertainty of position or time estimation of the receiver, by a smaller or a bigger amount. The DOP figure is a multiplier to the already existing uncertainty value, thus yielding the total uncertainty after introducing the relative satellites position factor. The receiver can calculate internally the various DOP figures so as to track the most favorable satellite combination. In our case we have selected the Position DOP as this criterion.
- **dophys : 10.0**, sets the PDOP threshold above which the receiver will switch from 2D positioning to no positioning.
- **dopmask : 10.0**, sets the PDOP threshold above which the receiver will select a new set of satellites.
- **dopthr : 10.0**, sets the PDOP threshold above which the receiver will switch from 3D positioning to 2D positioning. The value that we selected for the above PDOP parameters is arbitrary. The value of ten is a relatively high value yielding inaccurate results. This value was selected though so that the receiver is given enough uncertainty tolerance that it doesn't change satellite combinations continuously, in order to achieve a small value of PDOP. Each time the satellite combination changes, an undesirable discontinuity in the receiver's fix coordinates is observed.
- **ephhold : 0**, tells the receiver to acquire the latest ephemeris data, whenever available.
- **epoch : 0.0**, tells the receiver that no time shift of its calculated outputs is needed.
- **fix : n**, specifies that the receiver will track six satellites if available, in order to calculate its outputs. The other option is to track the best four satellites combination, i.e. the minimum number required for a position fix. Our choice is based on the fact that the loss of tracking of one satellite, "upsets" the solution less in the first case.
- **ion : e**, enables the ionospheric correction model.
- **mask : 10**, make sure that the minimum elevation angle at which the receiver will attempt to track satellites is at the default value of 10.0.
- **ph : d**, to make sure that the receiver does not operate in differential mode.

- **pos : 1**, with this command we actually tell the receiver to output the Position /Status /Data message ("@@Ba ..."), once per second. The GPS program software internally creates the required "@@BamC<CR><LF>" message that is described in page 4-80 of Ref.2, and sends it to the receiver.
- **rng : 1**, the same procedure as above followed, the receiver now also outputs the "@@Bg ..." message once per second.

It turns out, however, that not all the receiver's messages can be requested with the evaluation software kit. This is the case for the third message that we need: the Position /Status /Data extension message ("@@Bk..."). In order to have it included in the receiver output we have to develop our own executable program that will send the receiver the command character string "@@BkmC<CR><LF>". This string is sent via the AC100 model C30 computer, by using the AC100 environment, and, in particular, the "user_ser.c" user interface to the IP-Serial Device Driver. This is done with the "user_SERIAL_out" function in a way similar to that of the "user_sample_SERIAL_in" function. Detailed description on how to use this function are given in pages 136-139 in Ref.5.

This user_ser.c file, (Appendix D), is completely unrelated to the main project named "gps". It is executed by itself, and resides in a project folder named "bk", with the sole purpose of sending the GPS receiver the above mentioned command character string. What we have to do next is to unhook the receiver's serial cable from the PC, and to hook it to the IP-Serial module, channelA of the C30. Next step is to run the "bk" project for a few seconds to make sure that the receiver gets the command string.

By now we have initialized the GPS receiver, defined critical operating parameters, and forced it to output the three messages that we are interested in, at the rate of one Hz. The receiver will keep this configuration in its non volatile internal memory, so that it will behave the same each time it's powered up. A very simple way to check if it outputs the correct messages is to hook

to a serial port of a PC. By entering Windows and running the Terminal program we should clearly see on the screen the headers "@@Ba", "@@Bk", and "@@Bg" followed by meaningless symbols. These latter symbols appear because Terminal fails to translate the complicated Motorola Binary Format byte stream into ASCII characters.

2. Configuring The Reference Receiver

The reference receiver's initialization procedure is exactly the same as for the UAV receiver with the following differences:

- instead of requesting the three mentioned output messages of the UAV receiver, we request the output of differential corrections by using the **corout : 1** command from the evaluation software kit.
- We also tell the receiver to hold its position in order to extract differential corrections using the **ph : e** command from the same environment.
- We specify the position hold coordinates of the reference receiver, i.e. the receiver's position, by using the command **php (lat lon hgt [gps|msl])**, as specified in page 36, Ref.7.

The reference receiver will also keep these settings after powered off, as they are registered in its internal non volatile memory.

B. HARDWARE CONNECTIONS

As soon as the two receivers are initialized and configured, they are ready for the differential setup. The UAV receiver's serial I/O port is hooked to the IP-Serial module number 3, channelA, of the AC100 model C30, through a special adapter cable. The GPS receiver's serial port is interfaced using a standard nine pin serial connector. Pins 8,9 and 10 of Fig.4.3 are accessed via the pins 2, 3, and 5 of the serial connector, representing the RS-232 transmit, (TXD), receive, (RXD), and ground, (RTN), signal levels. The equivalent pins of the 50 pin IP-Serial connector are shown in Fig.4.8.

The reference GPS receiver can be either connected to the C30, for outputs to be monitored and recorded, or it can be connected directly to the UAV receiver by means of wires, providing the latter with differential capability. In the first case the connections follow the same rules as in the paragraph above. The only difference is that instead of channelA, it should be connected to channelB. In the second case, pin 2 of the reference receiver's nine pin serial connector is connected to pin 3 of the UAV receiver's nine pin serial connector. We also make sure that pins 5 of both receivers are connected together, so that the two devices share the same ground.

During the actual implementation of the DGPS setup, all wired connections leading to the UAV receiver will be substituted by a radio link. The only requirements for the radio link are to operate at 9600 baud and to be transparent to the user.

VII. ON THE FIELD TESTING AND PERFORMANCE EVALUATION

The task that concludes this thesis is the actual positioning data collection and evaluation. The PVT-6 receiver was operated on DGPS and non DGPS mode, and the data was compared together. Finally, some thoughts on the DGPS aided navigation system setup are presented.

A. EQUIPMENT SETUP

As a full duplex data link was not available during this thesis work, both receivers were placed next to each other with the differential receiver providing pseudorange corrections via cable link. The antennas were also placed side by side. The location where the experiment was held did not have an unobstructed view of the GPS satellites, as trees blocked about one fourth of the sky. For this reason the differential capability was lost many times, yielding a degraded position accuracy.

Whether the UAV receiver was operating in differential mode, or not, could be verified by monitoring the value of the receiver status byte, which is part of the Position\Status message ("@@Ba ... "). The status byte at any instant indicates the receiver's mode of operation, for example Differential, 3DFix, Altitude Hold, etc., depending on the value of the byte. We first examined the non-differential positioning data.

B. NON DIFFERENTIAL POSITIONING DATA

Non differential positioning data was collected several times. A typical set of data covering a time period of about half an hour is presented next.

The data was gathered on Wednesday, May 24, at the Naval Postgraduate School golf course at a location of Latitude $36^{\circ}35'22''N$ (1317220 deciseconds) and Longitude $121^{\circ}51'49''W$ (-4387090 deciseconds).

Fig.7.1 shows the variation in Latitude. During the 1250 seconds of data collection, the amplitude of the latitude change was about 70 meters (shown here in deciseconds of a degree of latitude). Fig.7.2 shows the equivalent graph for the longitude. Again the amplitude was about 70 meters. Fig.7.3 shows the plot of latitude versus longitude. The height plot is shown in Fig.7.4. Here the amplitude was 440 meters. It is obvious that the non differential position information cannot be used by a flight controller due to its inaccuracy and great variations.

C. DIFFERENTIAL POSITIONING DATA

A big improvement in position accuracy was observed in the differential setup. Fig.7.5 shows the latitude values obtained during 1800 seconds of data collection. First, note that the receiver is operating in differential mode from 0 to 250 seconds, and then again from 1000 to 1800 seconds. This can be clearly seen by examining the receiver status byte (Fig.7.15).

The differential three dimensional position fix mode was not available from the PVT-6 receiver all the time. This is because the differential station would either not output corrections for all the satellites used by the second receiver, (status byte = 32 case, non differential three dimensional mode), or both receivers would track three satellites only, (status byte = 20 case, differential two dimensional mode). The loss of tracking in each case happened because the location of the experiment did not have unobstructed view of the sky resulting in satellite "loss" when the latter moved behind the trees present. This factor can be eliminated by choosing another location for

the differential station. The receiver onboard Archytas should have no problem as the vehicle will have good visibility while flying.

The variations in latitude is now plus or minus 3 meters. The plot for longitude reveals also plus or minus 3 meters variations (Fig.7.6). The plot of latitude versus longitude is shown in Fig.7.7. Here the axis have the same scaling, i.e., each tic mark represents one decisecond, or three meters. The dense part of the graph around the point (-4387088, 1317218) represents the data corresponding to the differential operation mode. The height data also shows great improvement, with variations of only plus or minus 5 meters (Fig.7.8).

The most interesting results are shown in Figs.7.9, 7.10 and 7.11. Here the x, y, and z ECEF coordinates are shown in meters. During the differential operation of the receiver the variation is plus or minus 25 centimeters for x and y coordinates, and plus or minus 1 meter for the z coordinate. This sub-meter accuracy deserves some attention. The question is why the position expressed in geodetic coordinate system has a plus or minus 3m variation, while when expressed in ECEF coordinates a 25cm variation is observed. This question could not be answered within this thesis timeframe, however one answer is obvious.

The coordinate system in which GPS internally expresses all positions (satellite's and receiver's) is the ECEF. In order to transform the ECEF coordinates into Geodetic ones, a non linear analytically unsolvable equation has to be solved. The position errors are amplified during the iteration process used to solve the latter equation. This one reason for the Geodetic position's decreased accuracy.

The velocity components in the Tangent Plane coordinate system are shown in Fig.7.12, Fig7.13, and Fig7.14. The amplitude here is plus or minus

0.1 meters per second around the 0.0 m/s true value (receiver is stationary) during the differential portion.

The last figure (Fig.7.15) shows the value of the receiver status byte during the 1800 second interval. The status byte has the value of 36 from 0 seconds to 330 seconds and from 1000 to 1800 seconds. According to Ref.2 that means that the receiver was operating in differential three dimensional position fix mode. In between, the status byte had the values of 32 or 20 which means that the receiver was operating in non differential three dimensional mode or differential two dimensional mode respectively, which explains the inaccurate results during this time interval.

Operating in differential three dimensional position fix mode, the PVT-6 gives results accurate enough to be used by the Archytas flight controller. The information to be provided to the controller is the ECEF position, which can be easily transformed into a Tangent Plane position, and the Tangent Plane velocity components. The reason for this preference is that the above information is ready to be blended with the inertial measurement unit outputs, as the latter are expressed in the Tangent Plane system as well.

If Archytas is meant to be flown far away from the differential station we should consider using a twelve channel parallel receiver at the base, just to make sure that differential corrections are available for whatever satellites the Archytas receiver is tracking.

VIII. CONCLUSION

This thesis task is to provide three dimension position and velocity information to the Archytas flight control computer, using Differential Global Positioning technique.

The GPS receivers used are two Motorola PVT-6 OEM's. The Motorola Binary Proprietary format was chosen as the communication protocol between the flight control computer, which is the AC100 model C30 by ISI, and the PVT-6, as well as between the receivers themselves. The main tasks of this thesis were the following three:

- To decode the GPS message (Motorola proprietary format) in order to have usable numerical results.
- To create the suitable environment into the AC100 modelC30 that will accept and process the GPS positioning information.
- To set up the actual DGPS system, collect data and verify that they are accurate enough to be used by the Archytas flight controller.

To reach the first objective, two C-code files were written. These files are shown in appendices B and C.

A model that reads seventy seven variables was built to reach the second objective. The variable's values are obtained from the GPS receivers through the Serial input-output IP module. After the data is decoded according to the previous paragraph, it can be recorded, analyzed, or directly fed into the flight controller.

Finally, the setup of the DGPS system was completed. The data collected showed an accuracy of plus or minus 3m (geodetic coordinates), to plus or minus 25cm (ECEF coordinates).

This level of accuracy permits the flight controller to blend the GPS and the Inertial Measurement Unit positioning data, so as to enhance its navigation problem solution.

APPENDIX A. : FIGURES

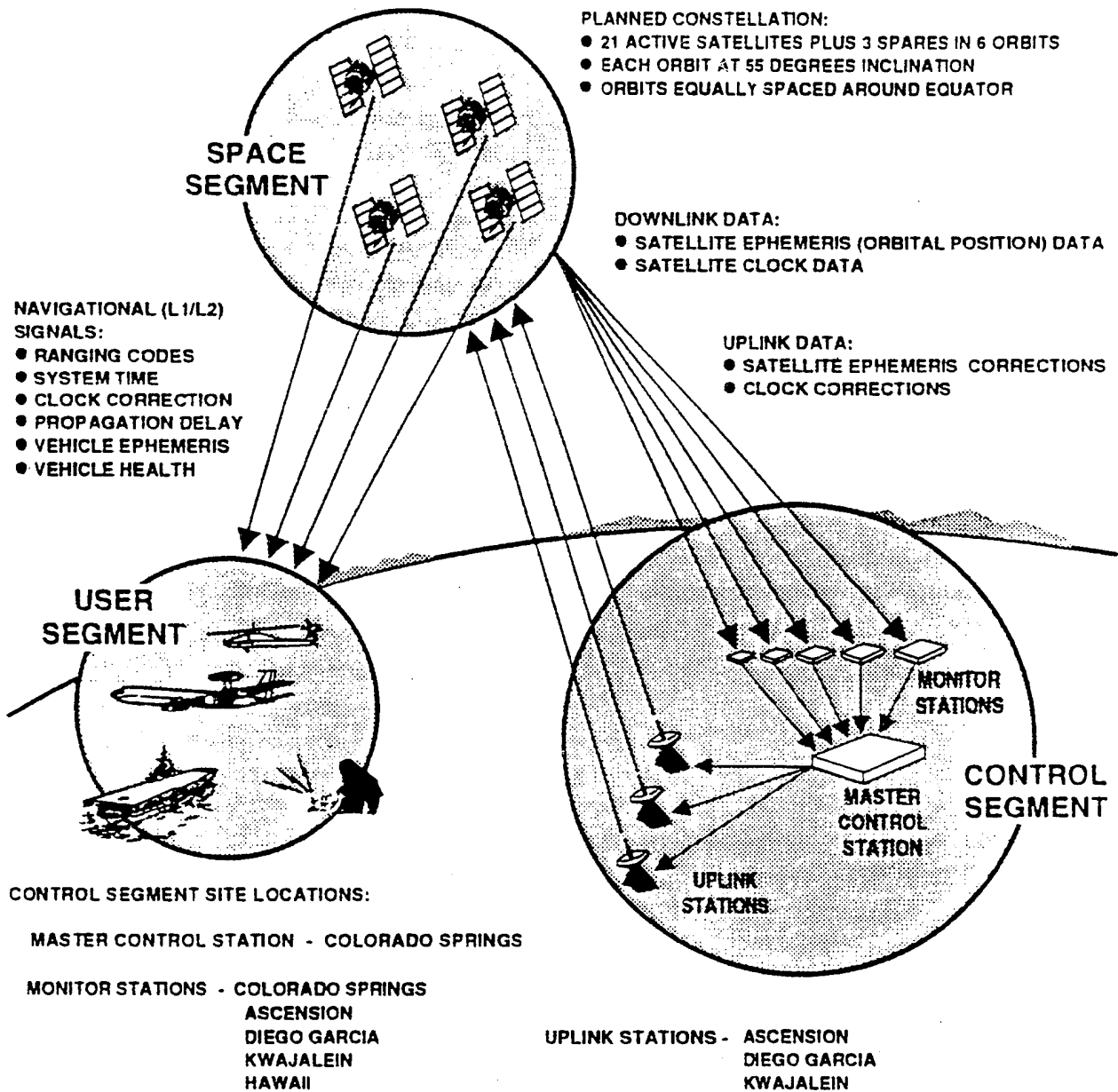
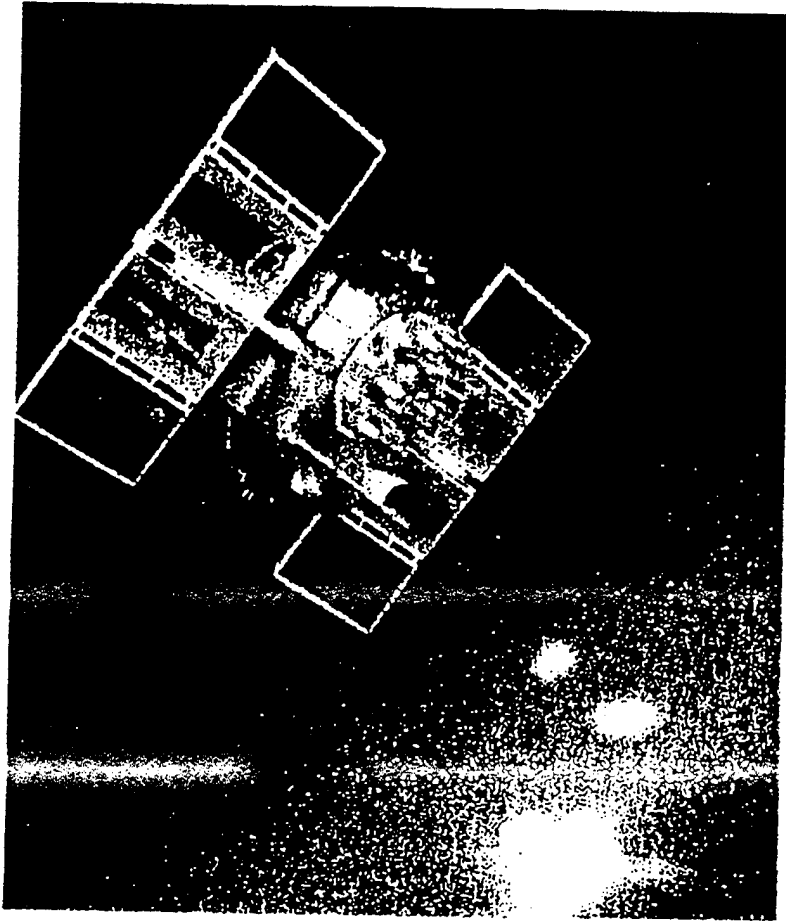


Fig.2.1 The GPS segments. From Ref.2



GPS Satellites

Name: NAVSTAR

Manufacturer: Rockwell International

Altitude: 10,900 nautical miles

Weight: 1900 lbs (in orbit)

Size: 17 ft with solar panels extended

Orbital Period: 12 hours

Orbital Plane: 55° to equatorial plane

Planned Lifespan: 7.5 years

Number built: 11 Block I prototype
satellites 28 Block II production
satellites

Constellation: 24 satellites

Fig.2.2 The GPS satellite. From Ref.8

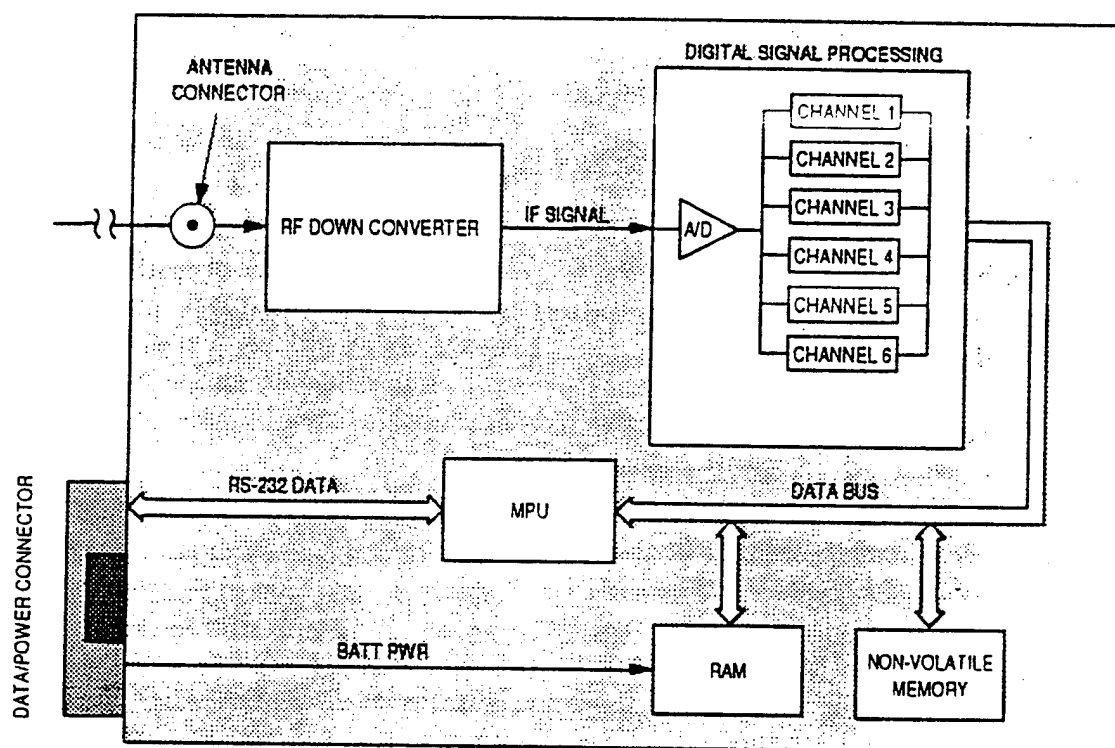


Fig.4.1 The PVT-6 receiver. Internal view. From Ref.2

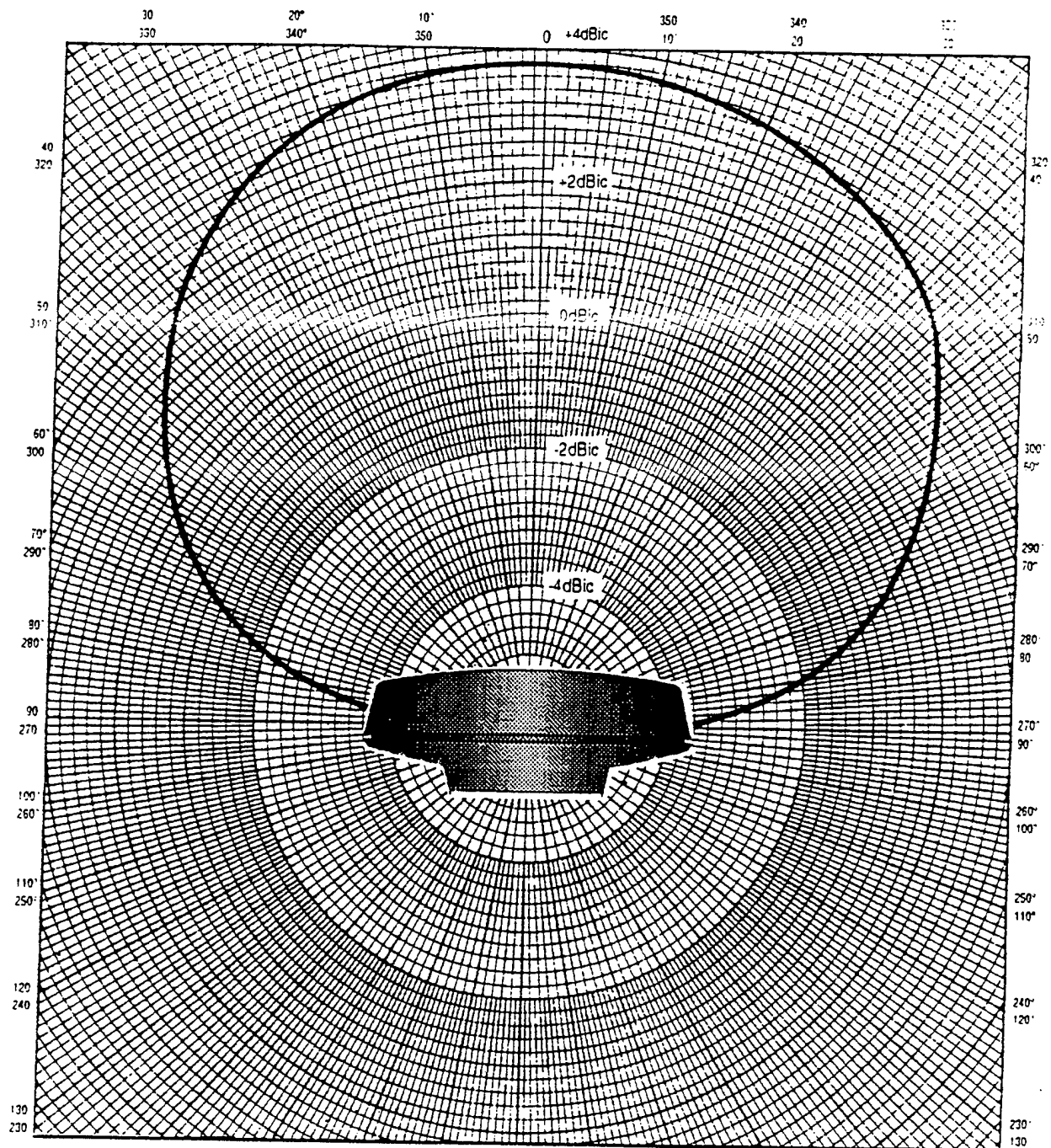


Fig.4.2 The PVT-6 receiver's antenna. From Ref.2

Pin Number	Signal Name	Description
1	+12V/+5V BATT	(Optional) +5 Vdc regulated or +12 Vdc unregulated for running Real Time Clock and retention of satellite ephemeris information stored in keep-alive RAM memory
2	+5V MAIN	(Optional) +5 Vdc regulated for power requirements of entire GPS Receiver (+12 Vdc signal not used)
3	+12V/+5V RTN	Power supply (+5V or +12V) return
4	Vpp ²	Flash memory (EPROM) programming voltage
5	+12V MAIN	+12 Vdc unregulated for power requirements of entire GPS Receiver
6	1 PPS ³	(Option A) 1 pulse per second output
7	1 PPS RTN	(Option A) 1 pulse per second return
8	RS232 TXD	Serial RS232 data output
9	RS232 RXD	Serial RS232 data input
10	RS232 RTN	Signal return for RS232 signals

¹ AMP connector PN 104369-4 (installed PCB); PN 103681-2 (mating)

² Not used (used only by Motorola for updating of software)

³ 1 Hz pulse train with rising edge coincident with UTC/GPS 1 second tick

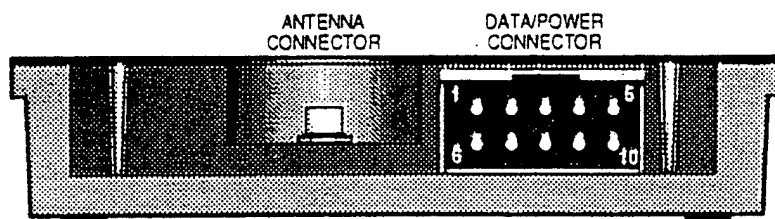


Fig.4.3 The PVT-6 receiver's serial port. From Ref.2

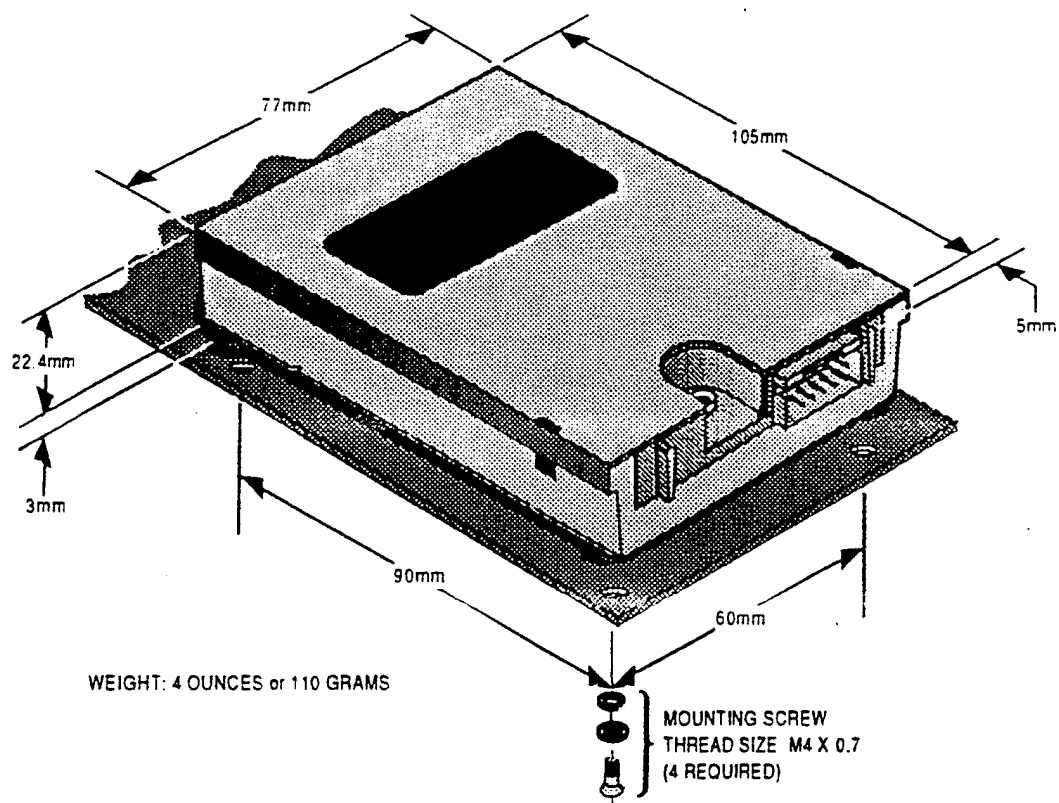


Fig.4.4 The PVT-6 receiver. From Ref.2

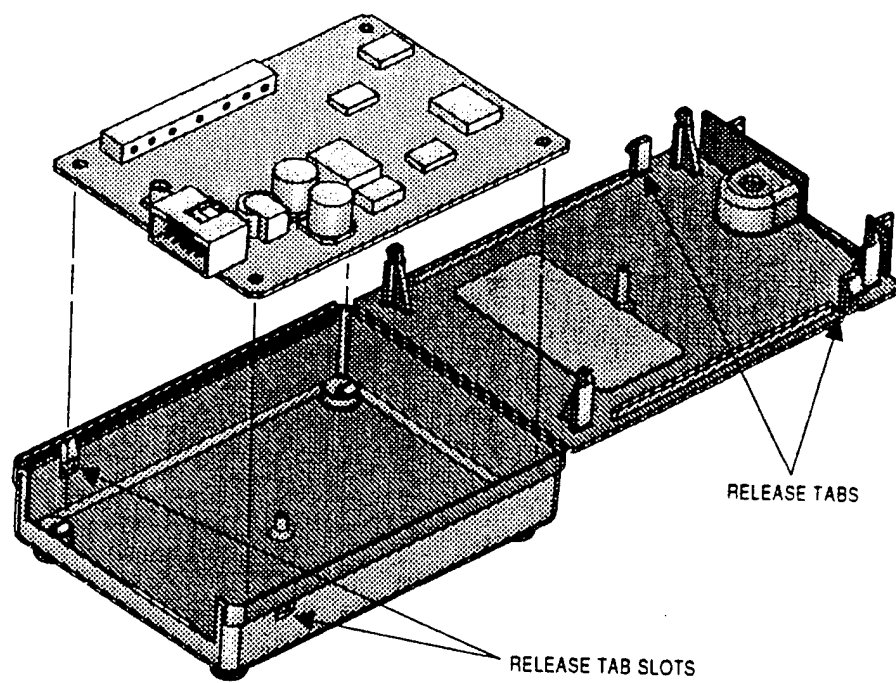


Fig.4.5 The PVT-6 receiver. From Ref.2

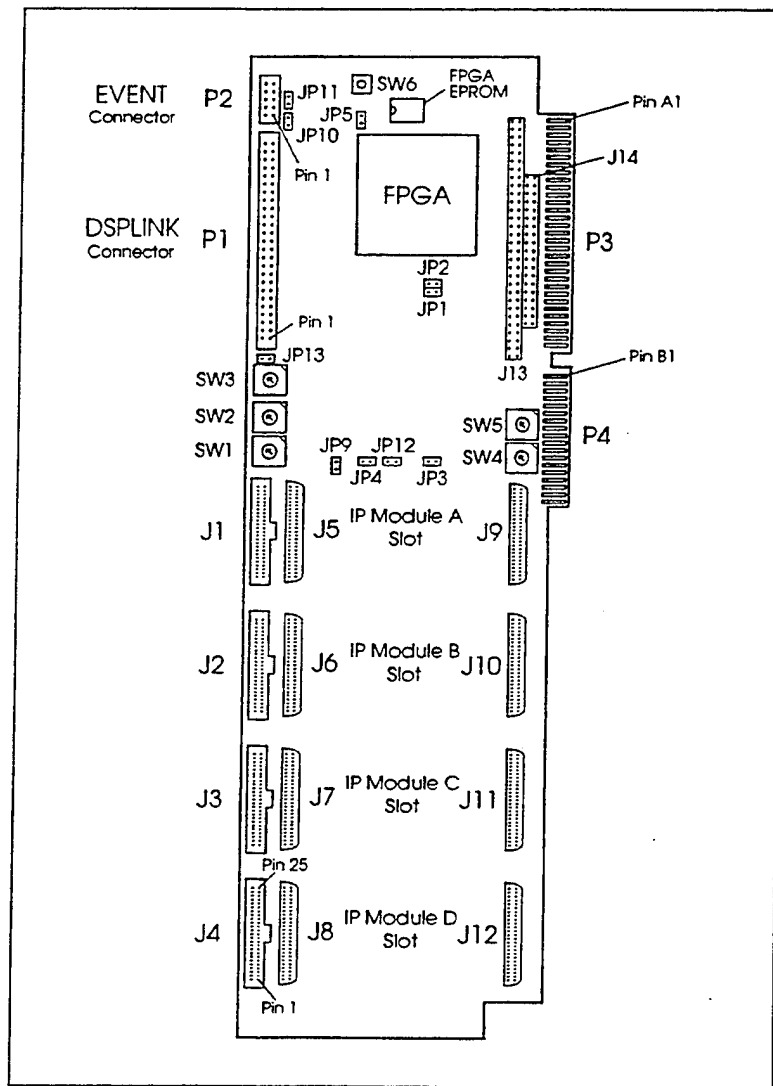


Fig.4.6 The DSPFLEX board. From Ref.2

DSPFLEX Bd 1		DSPFLEX Bd 2		DSPFLEX Bd 3		DSPFLEX Bd 4	
Position	IP #	Position	IP #	Position	IP #	Position	IP #
A	1	A	5	A	9	A	13
B	2	B	6	B	10	B	14
C	3	C	7	C	11	C	15
D	4	D	8	D	12	D	16

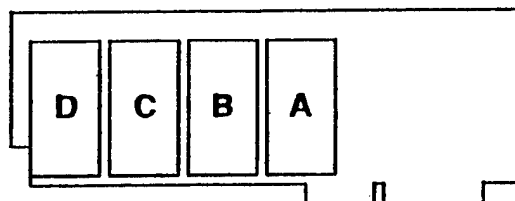


Fig.4.7 The IP modules. From Ref.2

CH-A Cable	CH-B Cable	D-shell (DB-25)	RS-232 Signal	RS-422 Signal
1	26	1	Ground	Ground
3	28	2	TxD	
5	30	3	RxD	RxD-
7	32	4	RTS	
9	34	5	CTS	CTS+
11	36	6	DSR/DCD	DSR+/DCD+
13	38	7	Ground	Ground
15	40	8	DCD	DCD+/RTXC+
17	42	9		TxD+
19	44	10		TxD-
21	46	11		DSR-/DCD-
23	48	12		DCD-/RTXC-
25	50	13		CTS-
2	27	14		DTR-
4	29	15		TRXC-
6	31	16		RxD+
8	33	17		DTR+
10	35	18		RTS+
12	37	19		RTS-
14	39	20		
22	47	24	DTR	TRXC+
24	49	25		+5 volt option

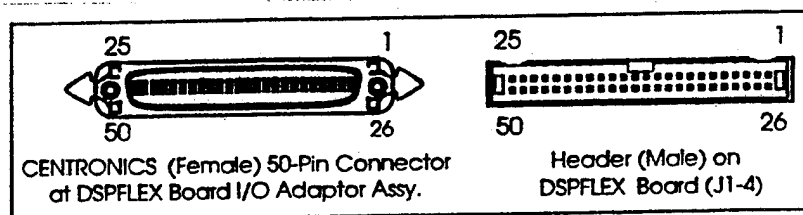


Fig.4.8 The Pin Out configuration of the IP Serial module. From Ref.5

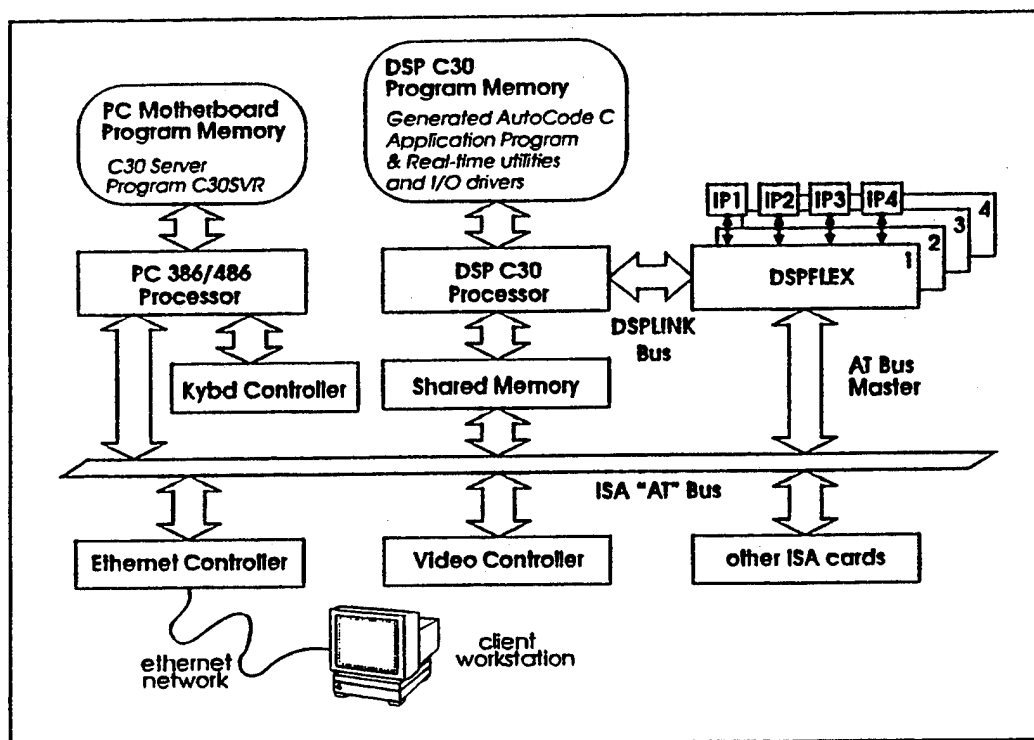


Fig 4.9 AC100 model C30 hardware setup. From Ref.5

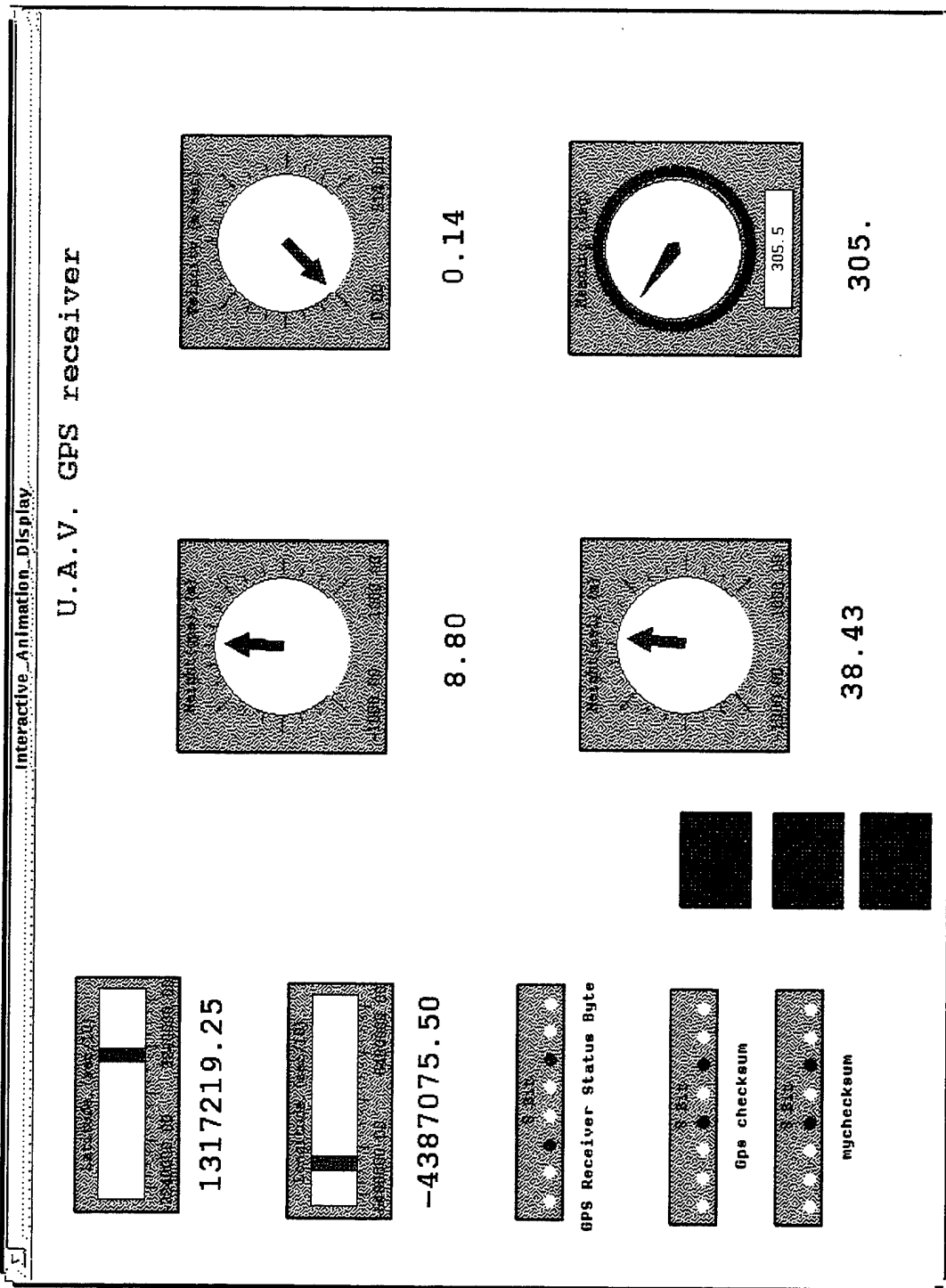


Fig.5.2 Latitude-Longitude-Height GUI screen

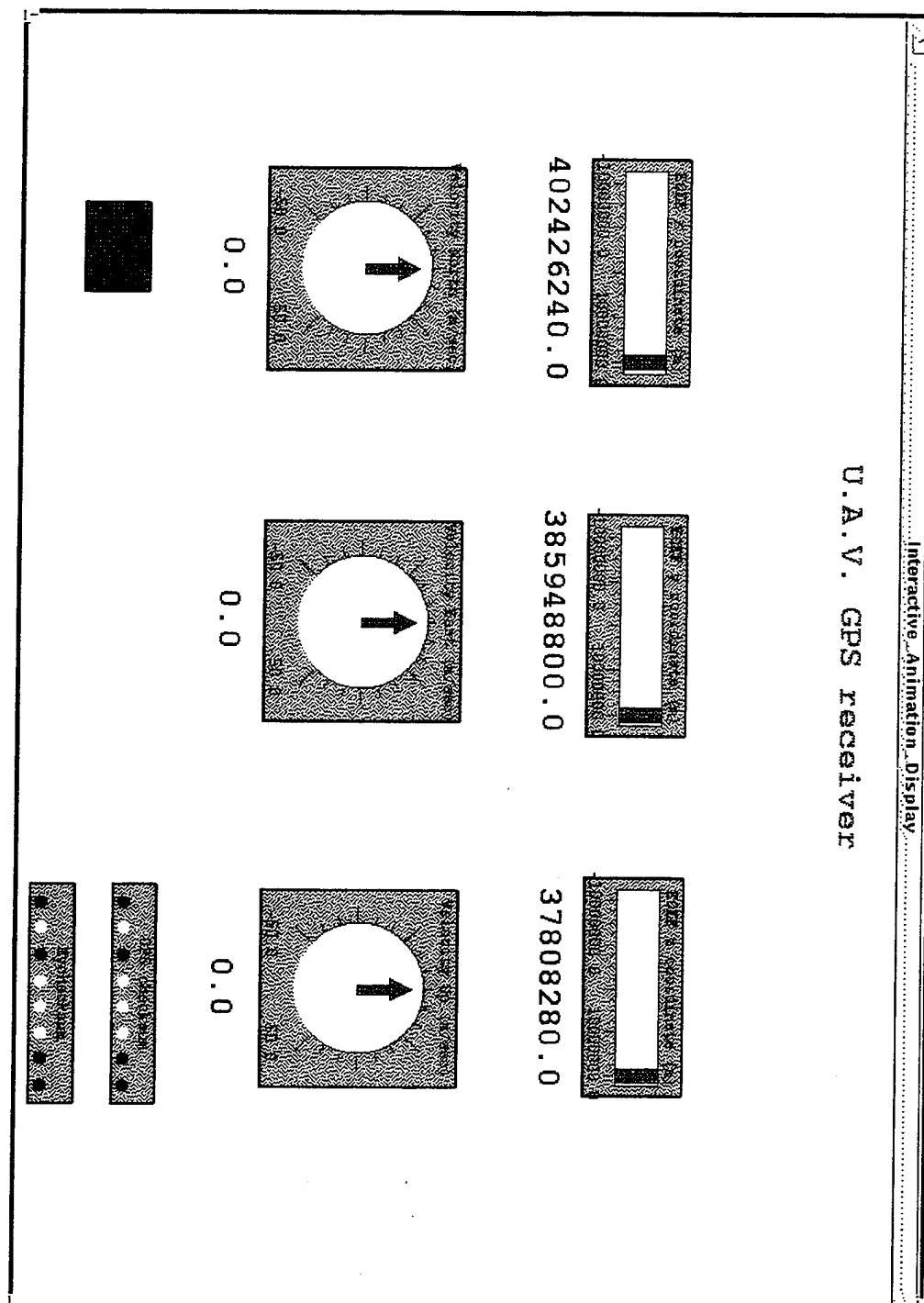


Fig.5.3 ECEF position, Tangent Plane velocity GUI screen

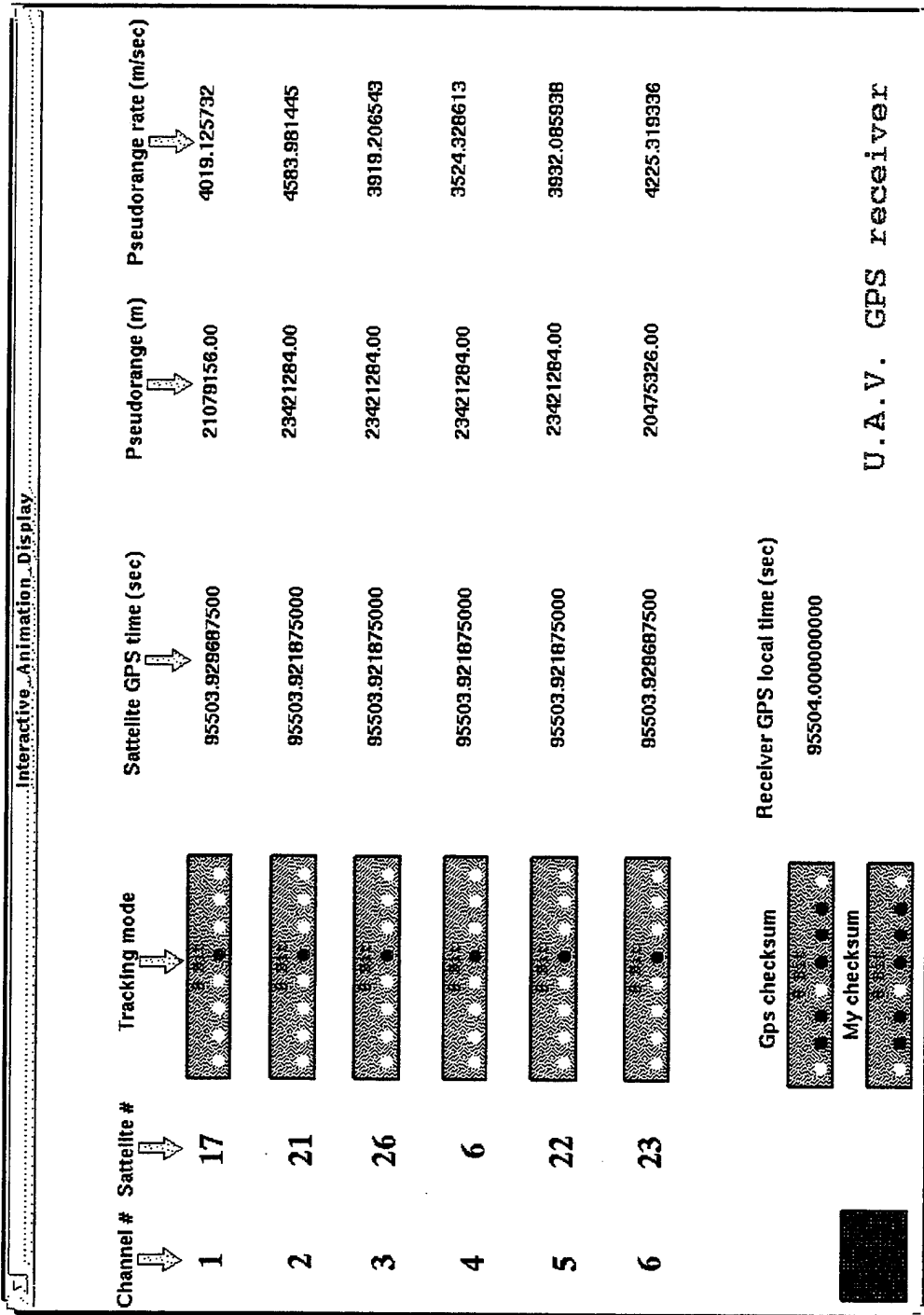


Fig.5.4 Pseudoranges GUI screen

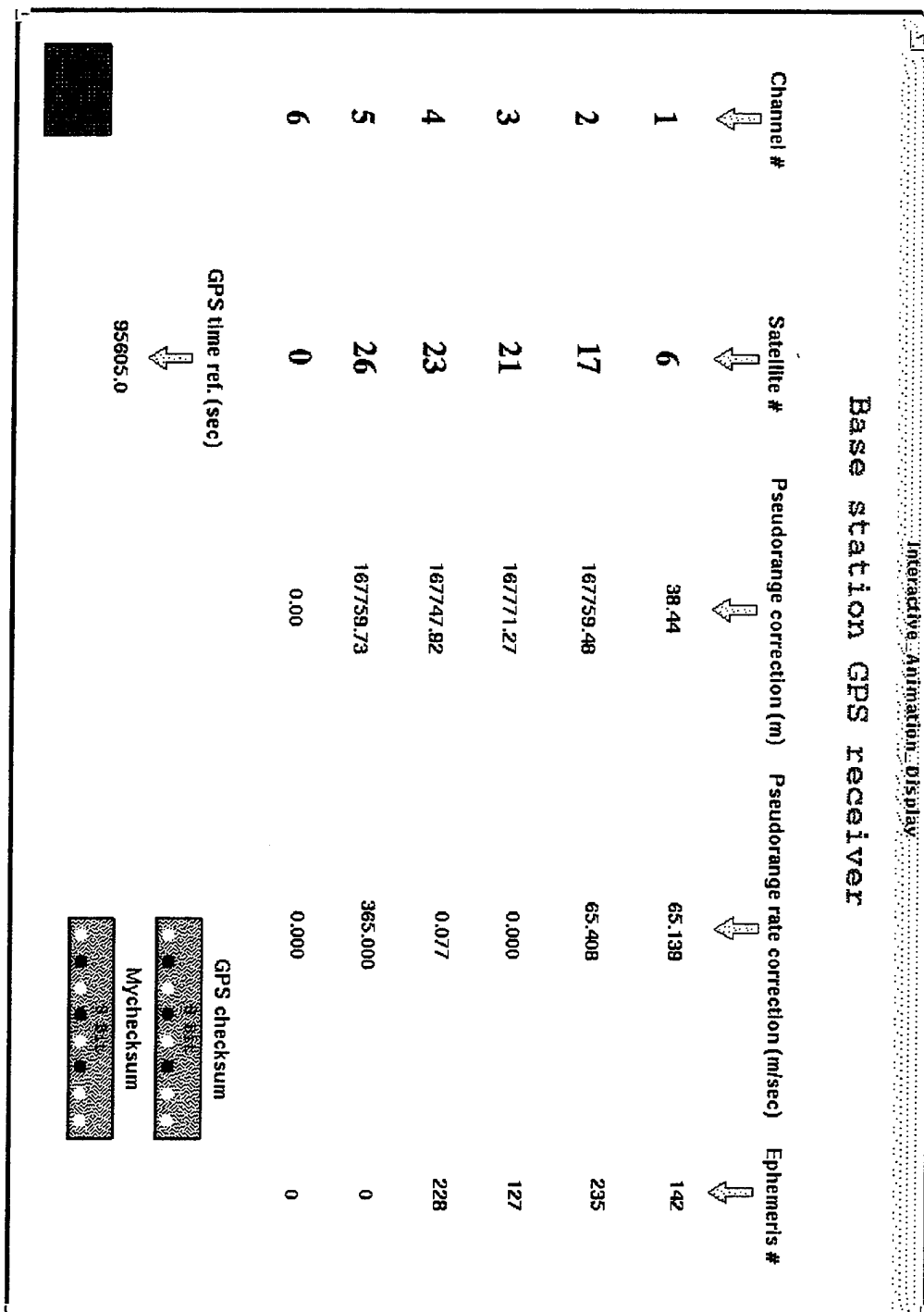


Fig.5.5 Differential corrections GUI screen

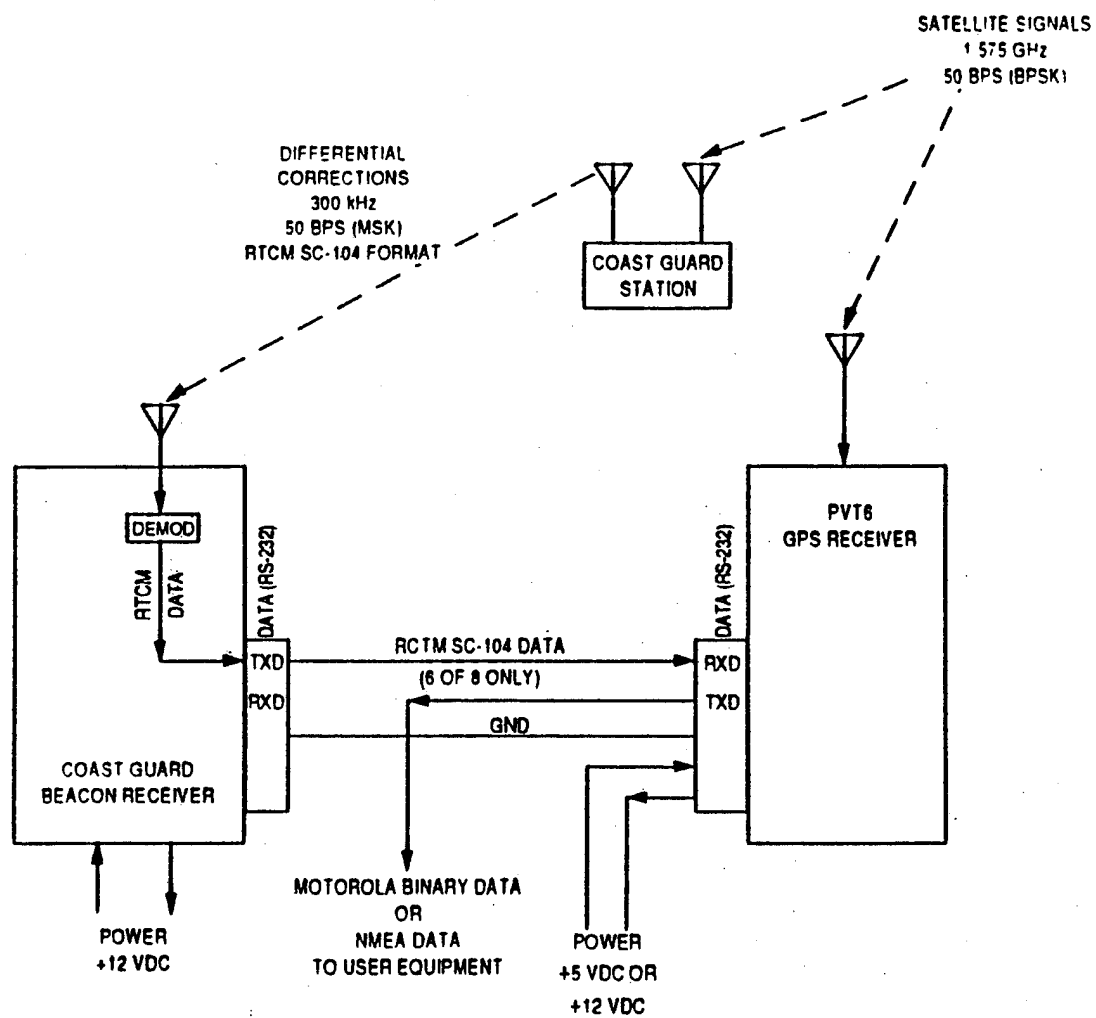


Fig.6.1 DGPS setup not requiring real time control. From Ref.2

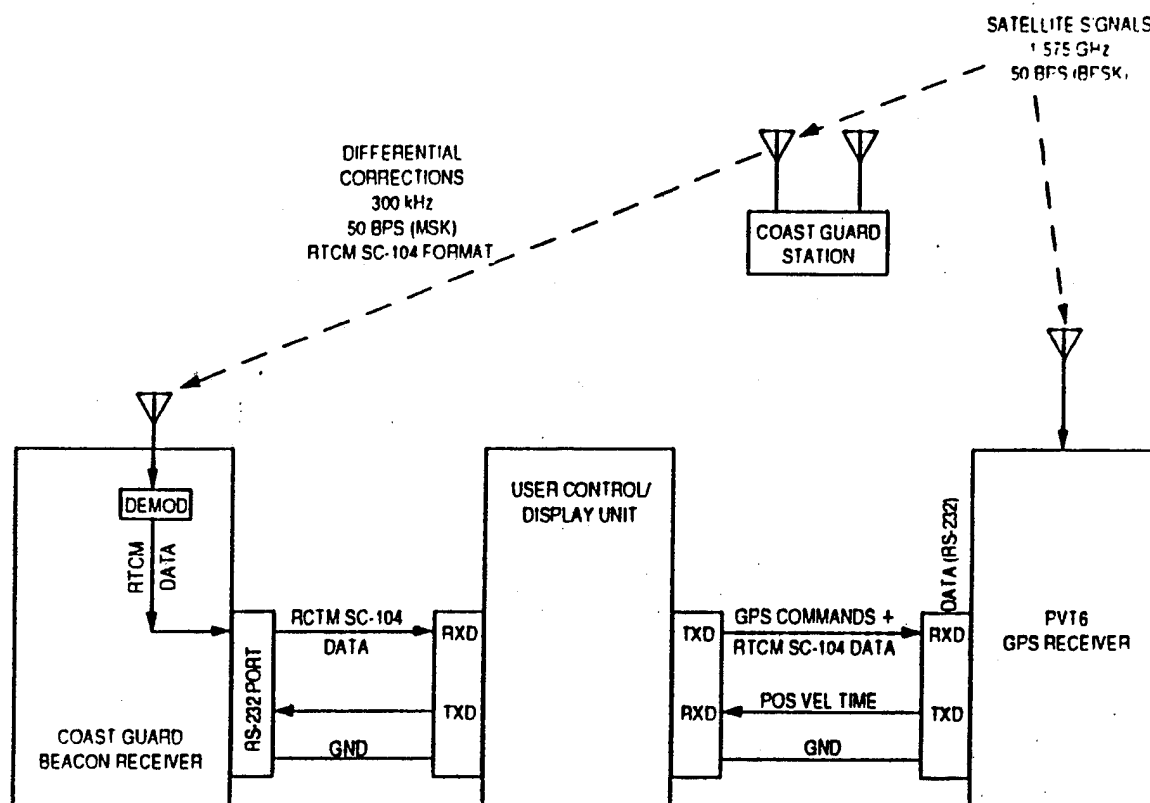


Fig.6.2 DGPS setup requiring real time control. From Ref.2

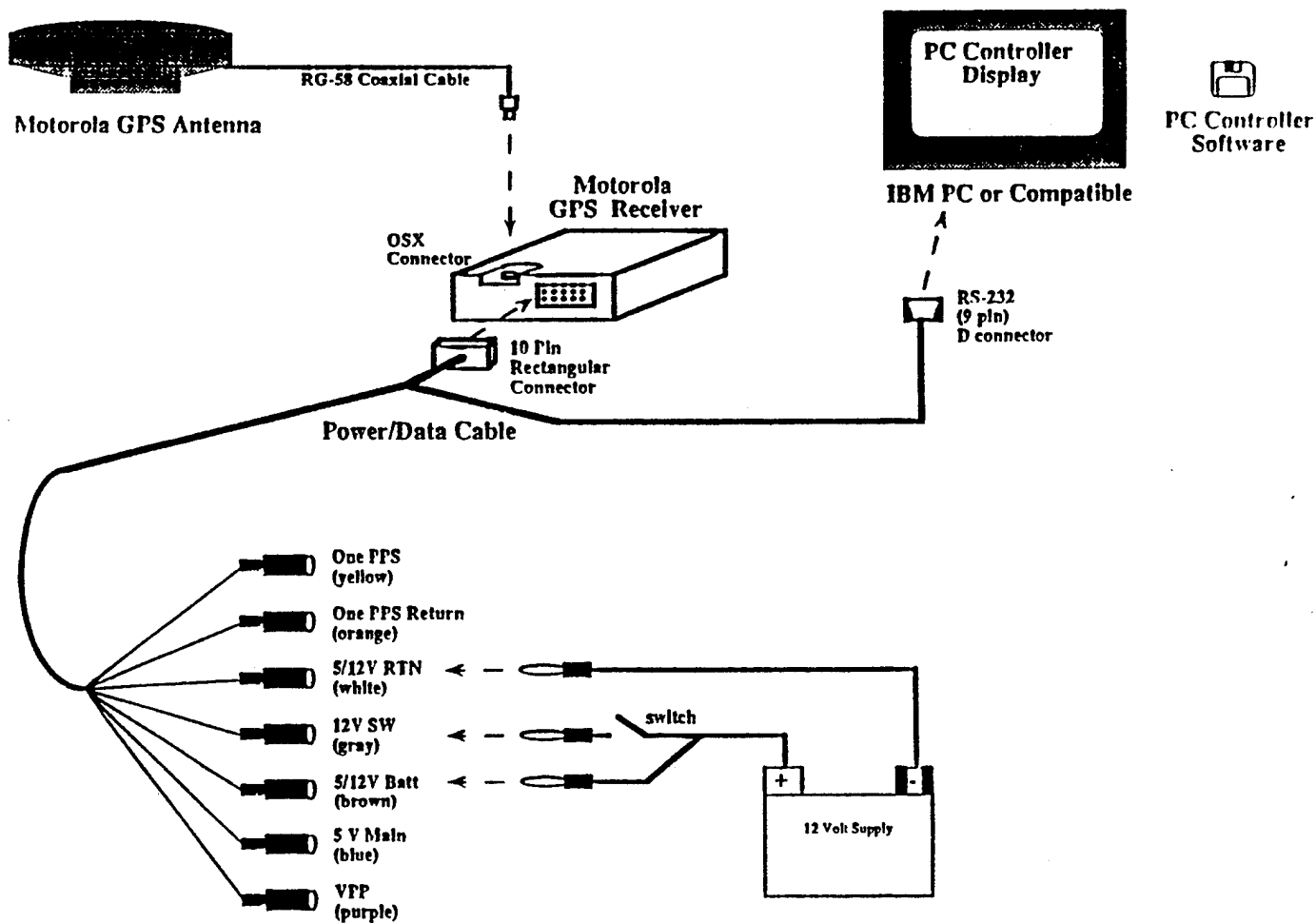


Fig.6.3 Configuration of the PVT-6 via the PC. From Ref.7

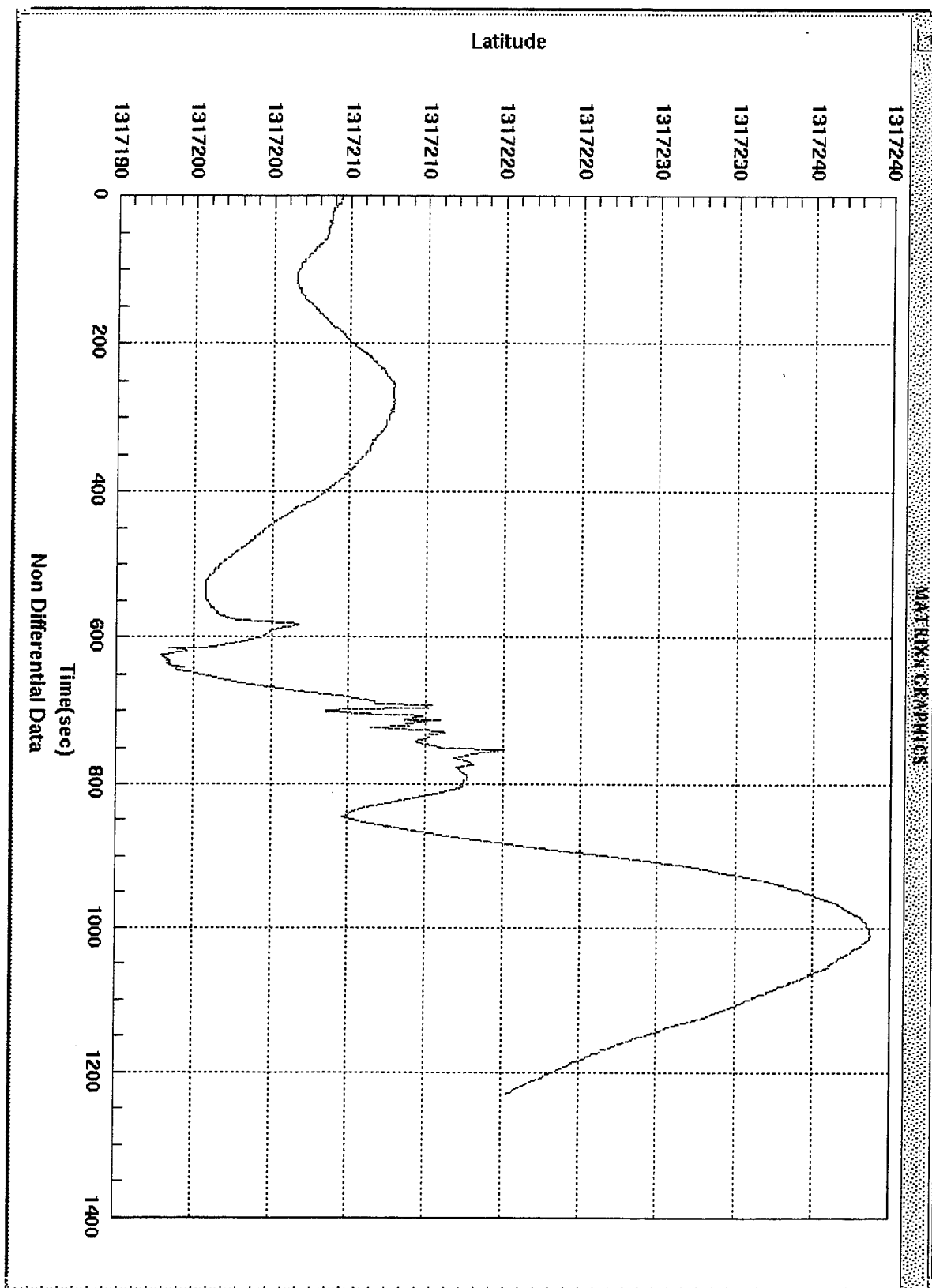


Fig.7.1 Latitude, non differential setup

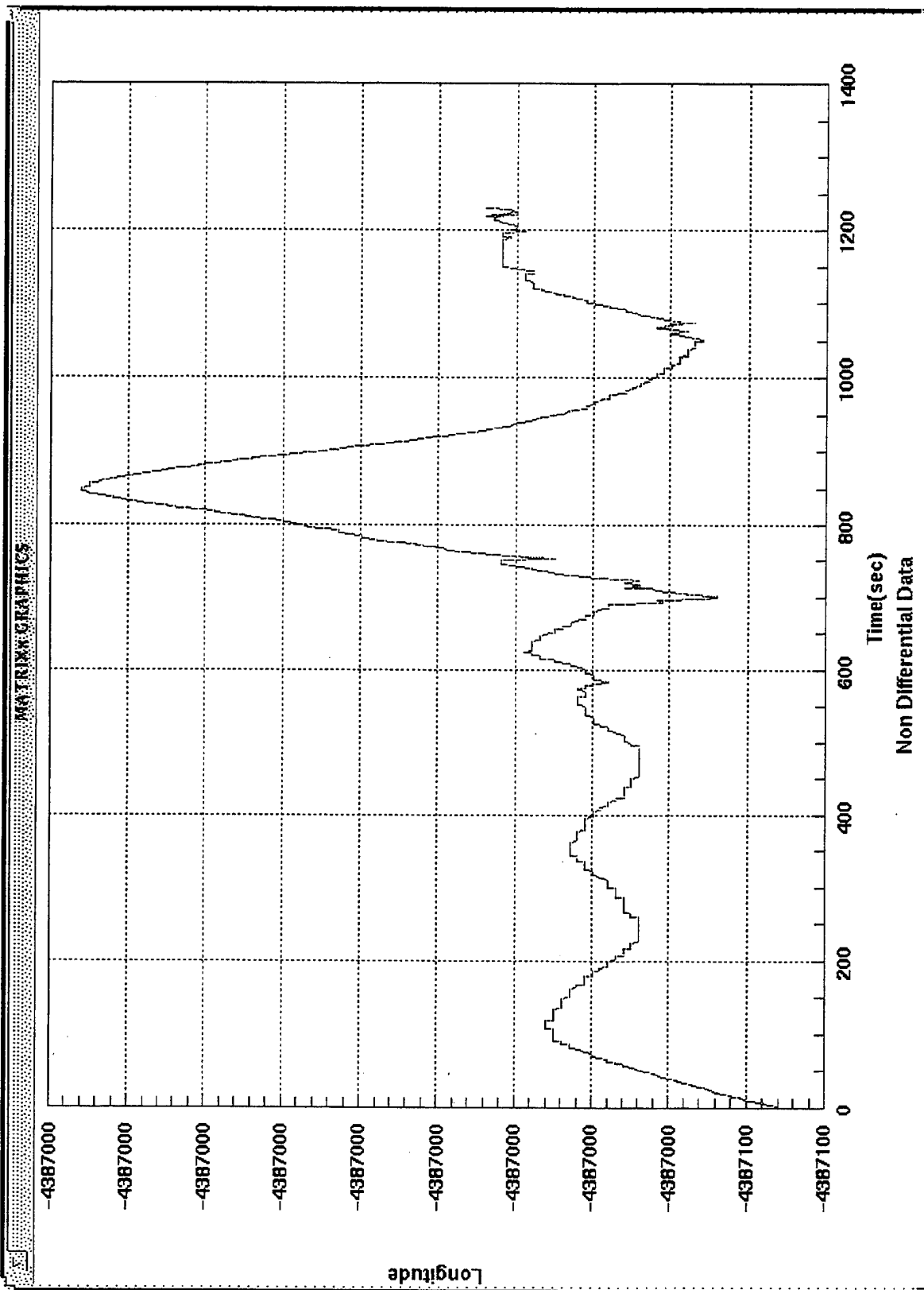


Fig.7.2 Longitude, non differential setup.

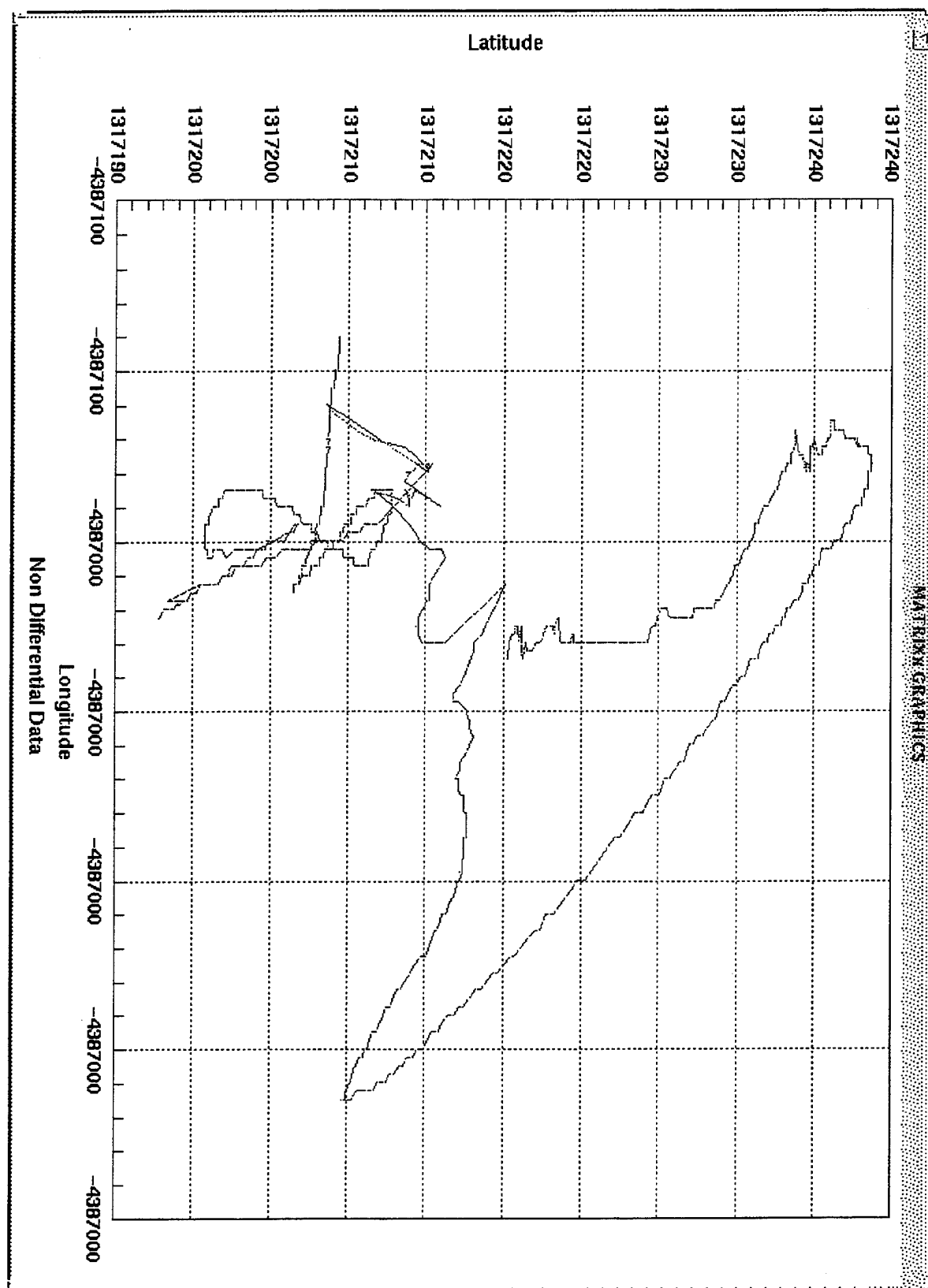


Fig.7.3 Latitude vs Longitude, non differential setup

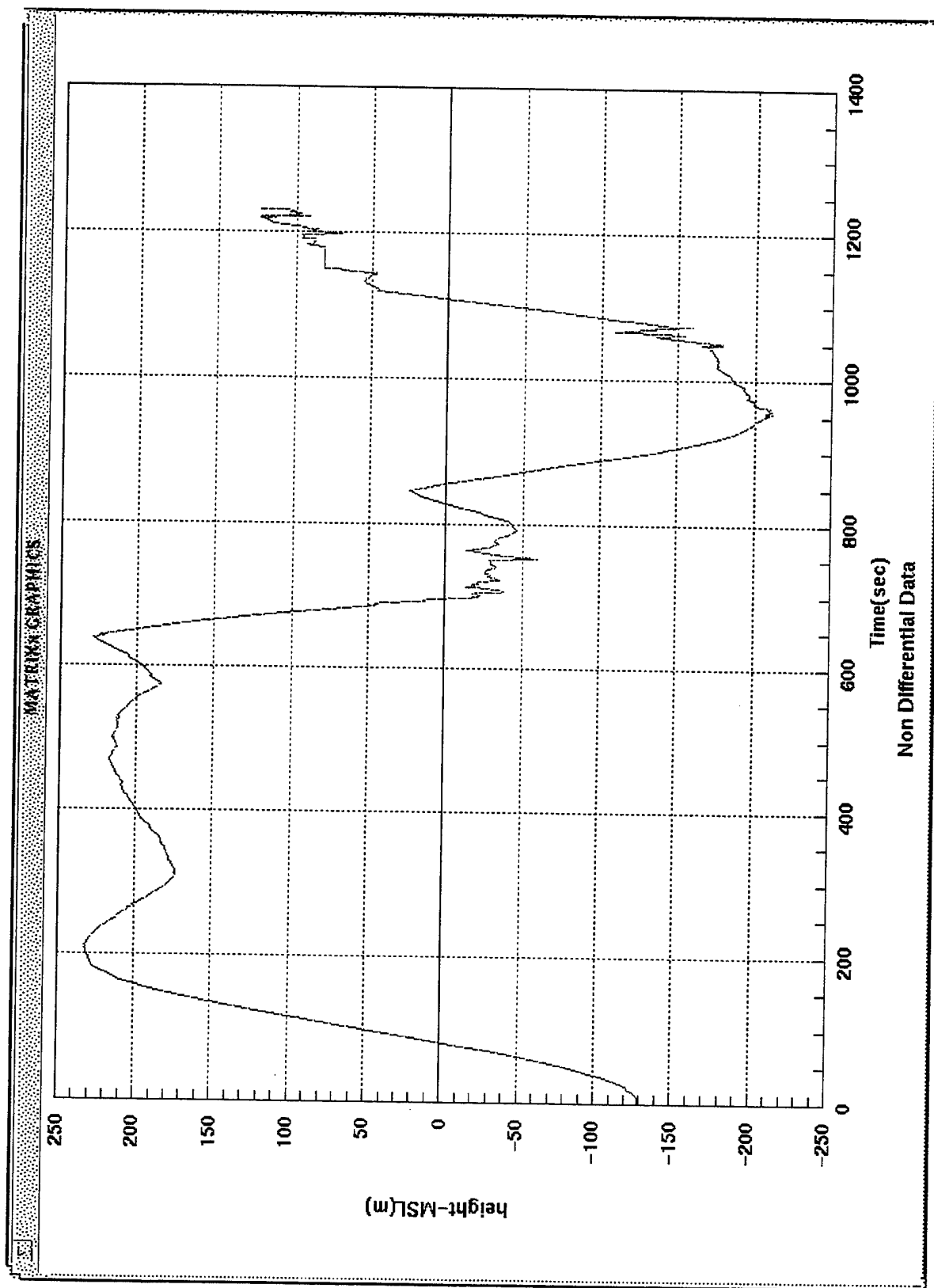


Fig. 7.4 Height, non differential setup.

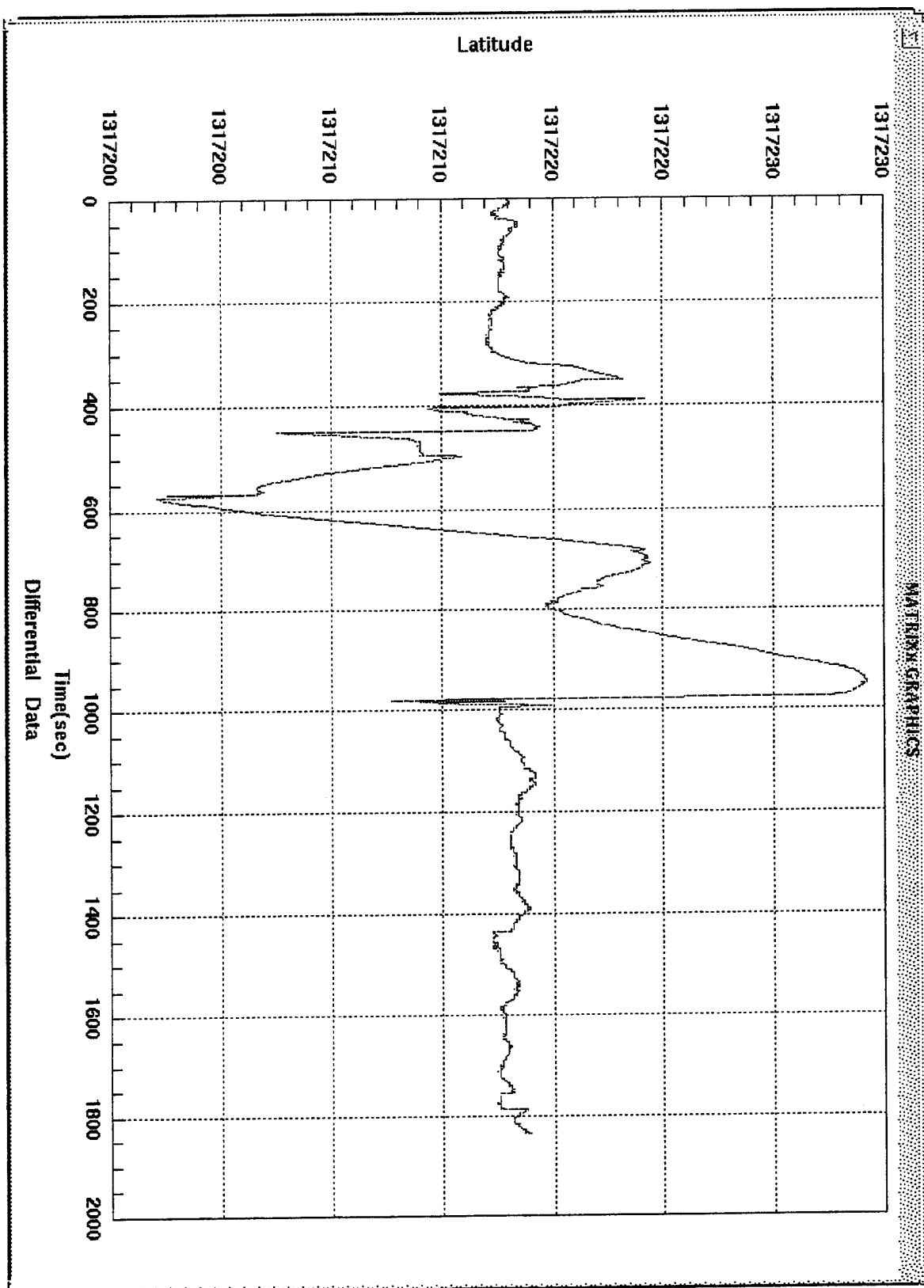


Fig. 7.5 Latitude, differential setup.

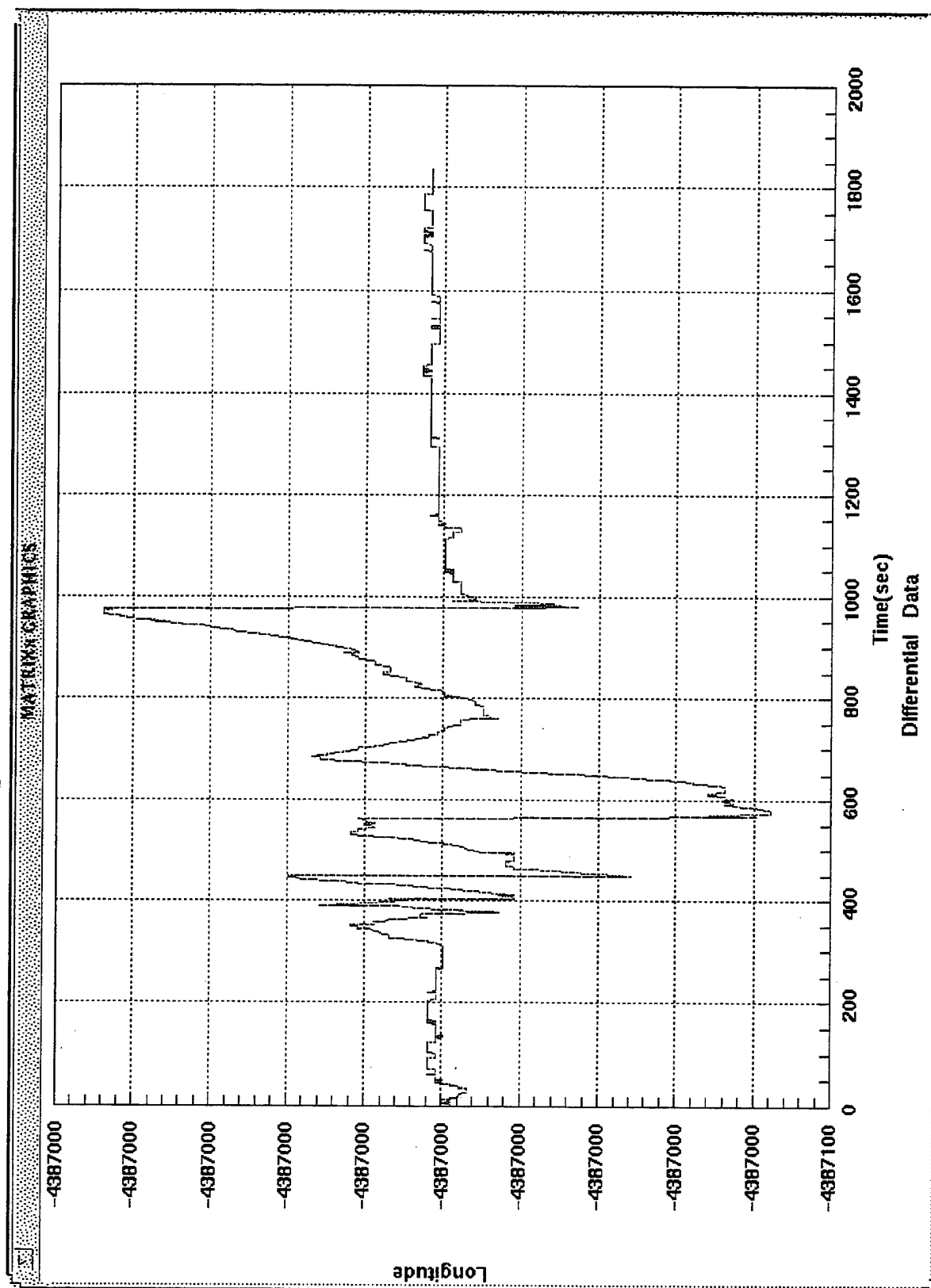


Fig.7.6 Longitude, differential setup.

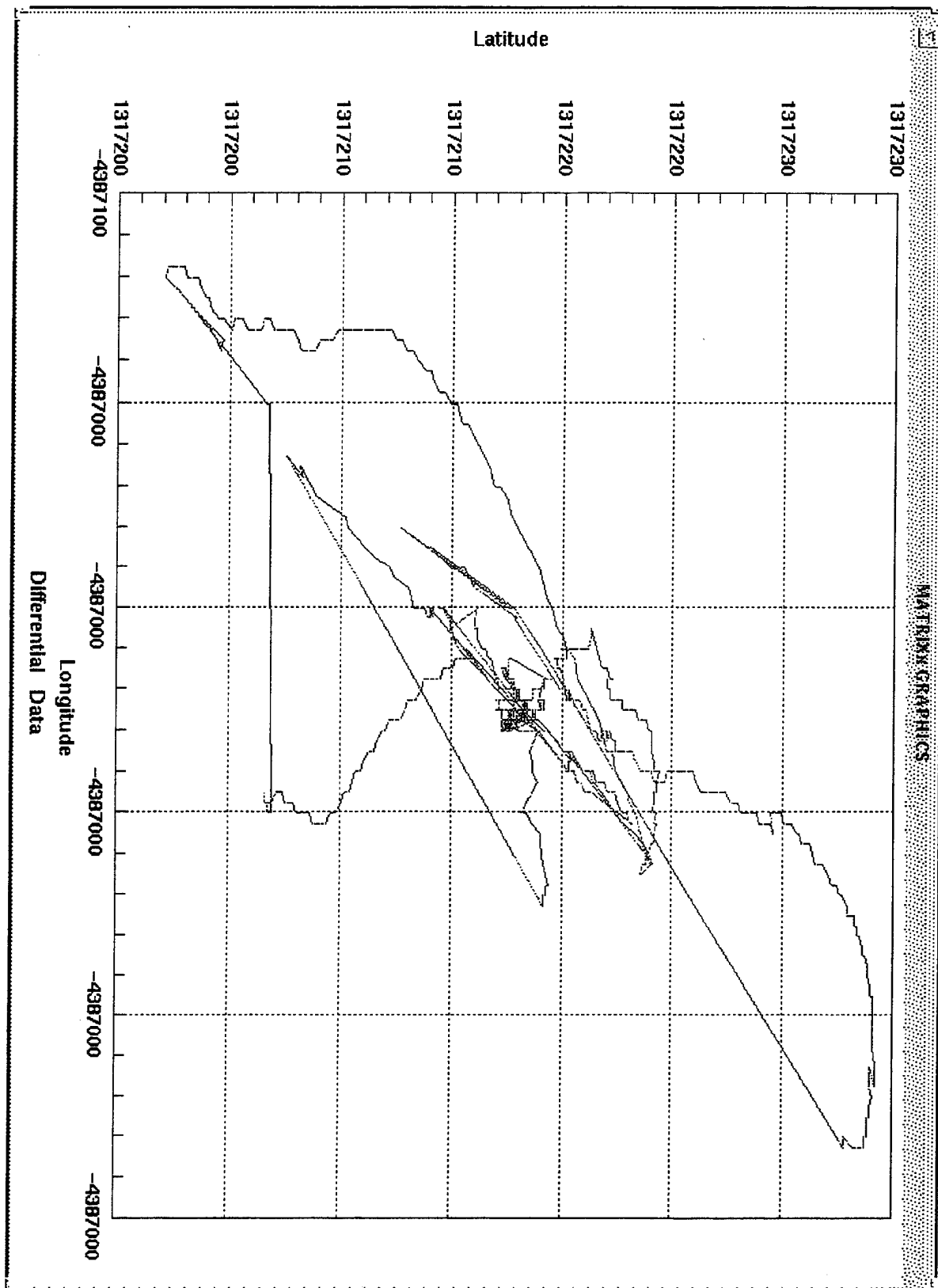


Fig.7.7 Latitude vs Longitude, differential setup.

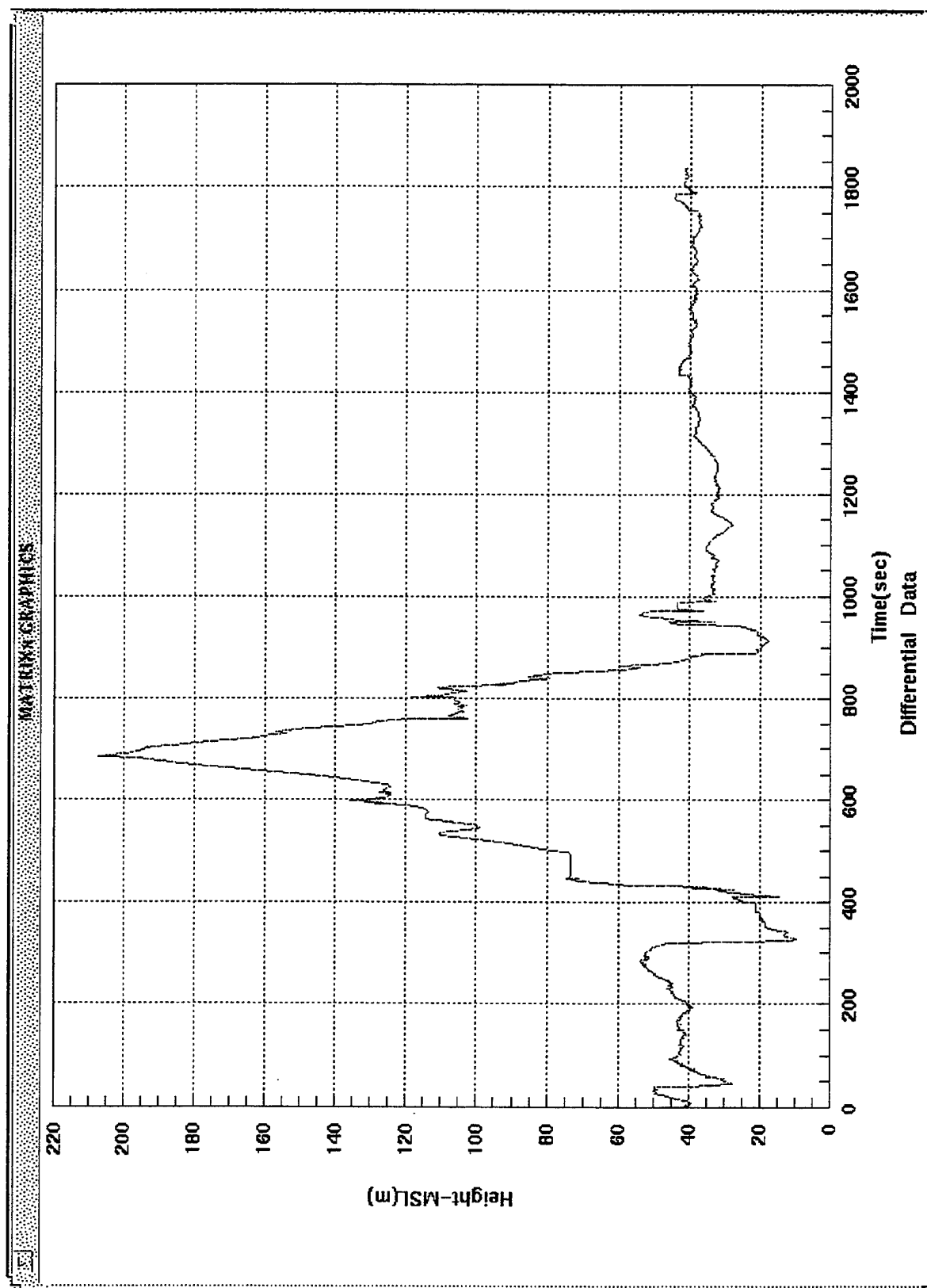


Fig.7.8 Height, differential setup.

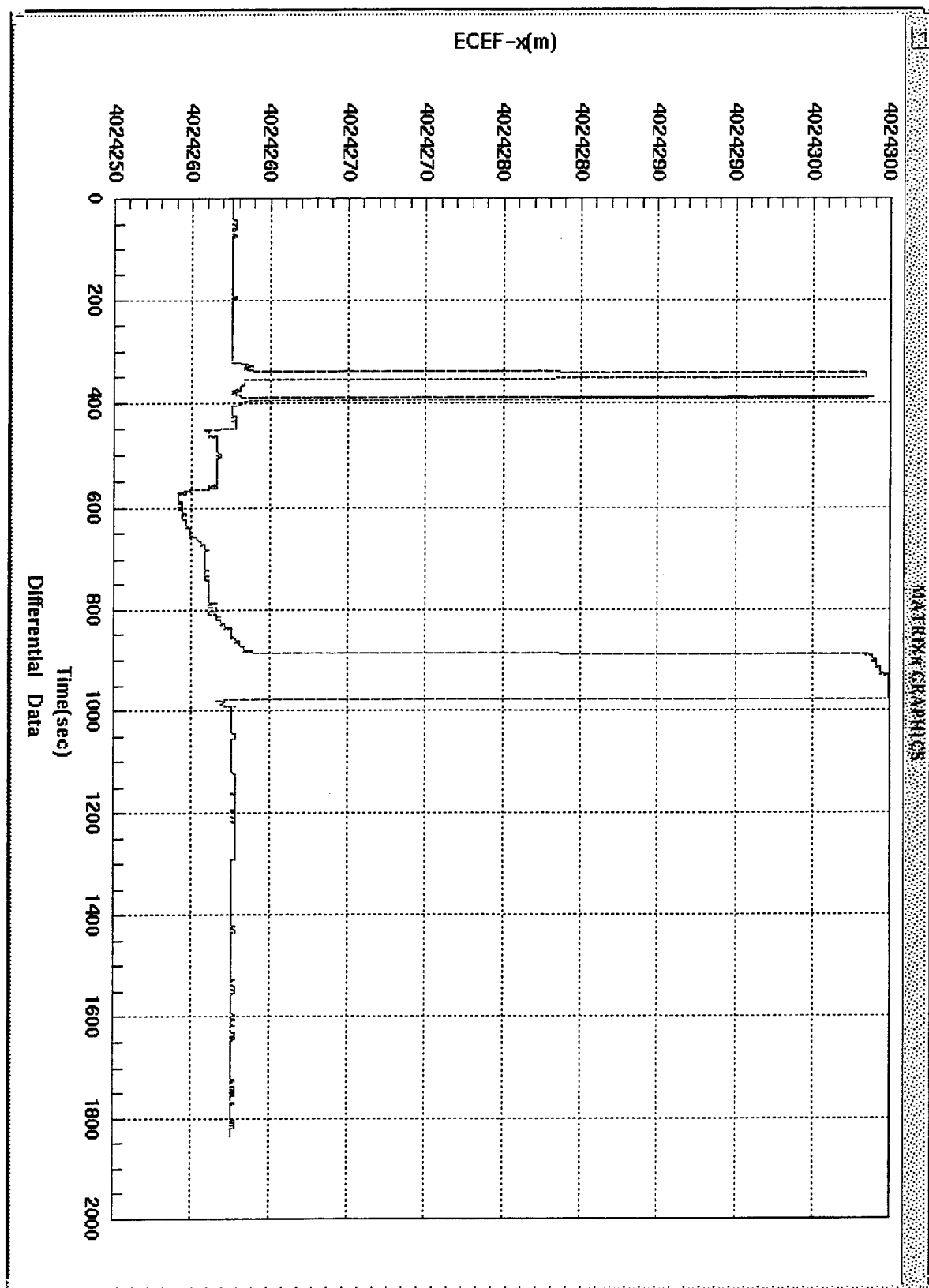


Fig.7.9 ECEf x coordinate, differential setup.

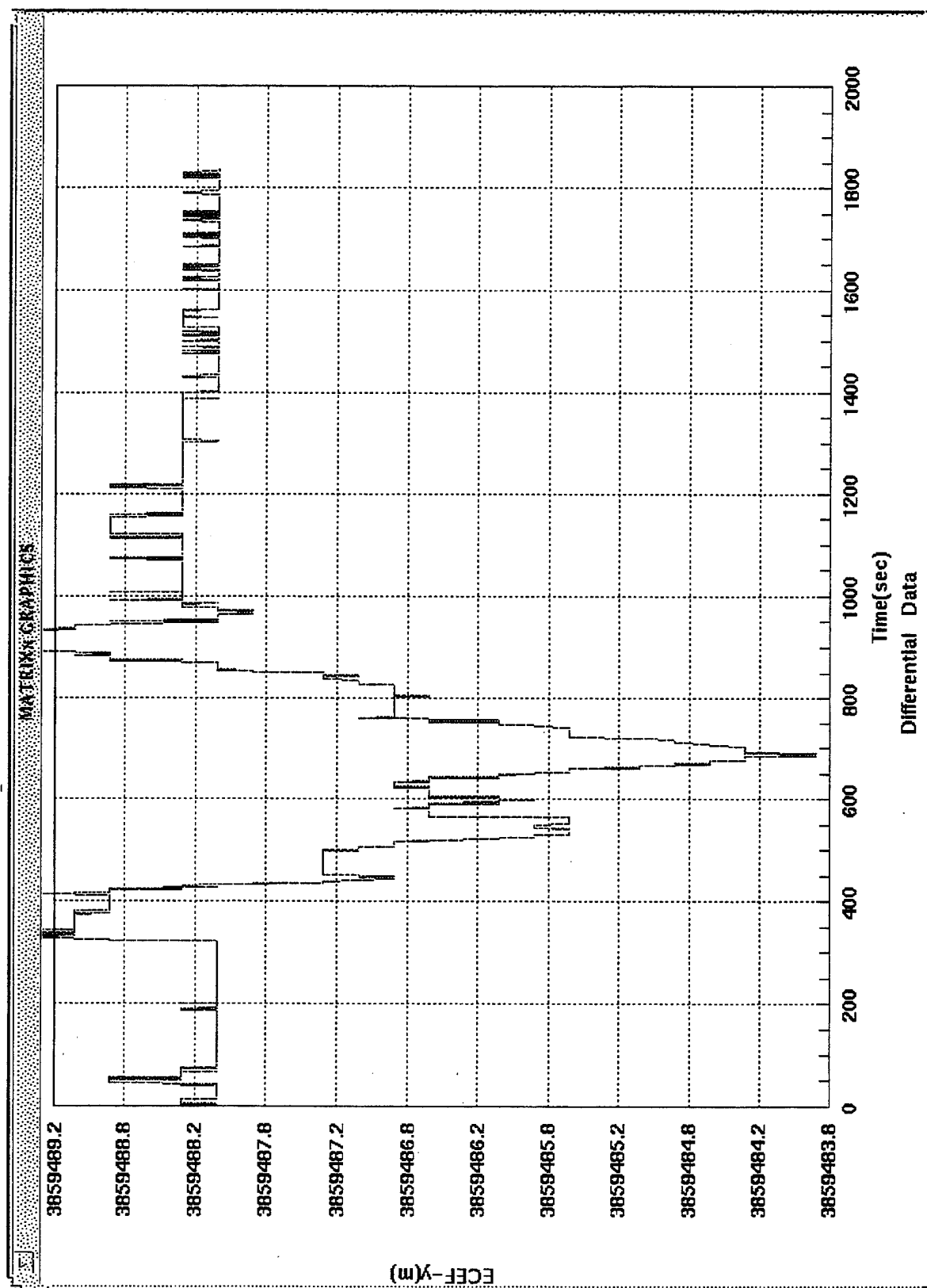


Fig.7.10 ECEF y coordinate, differential setup.

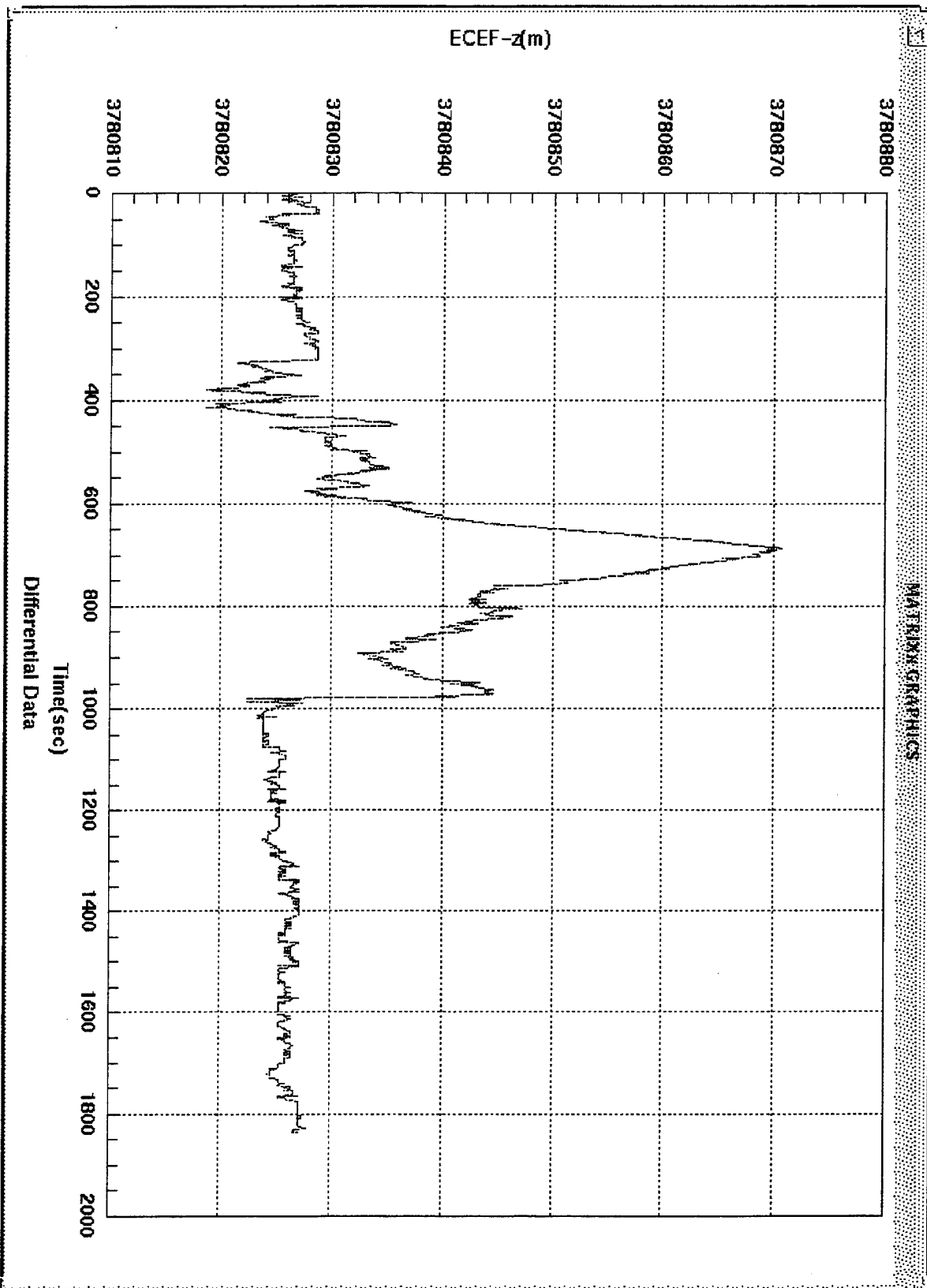


Fig. 7.11 ECEf z coordinate, differential setup.

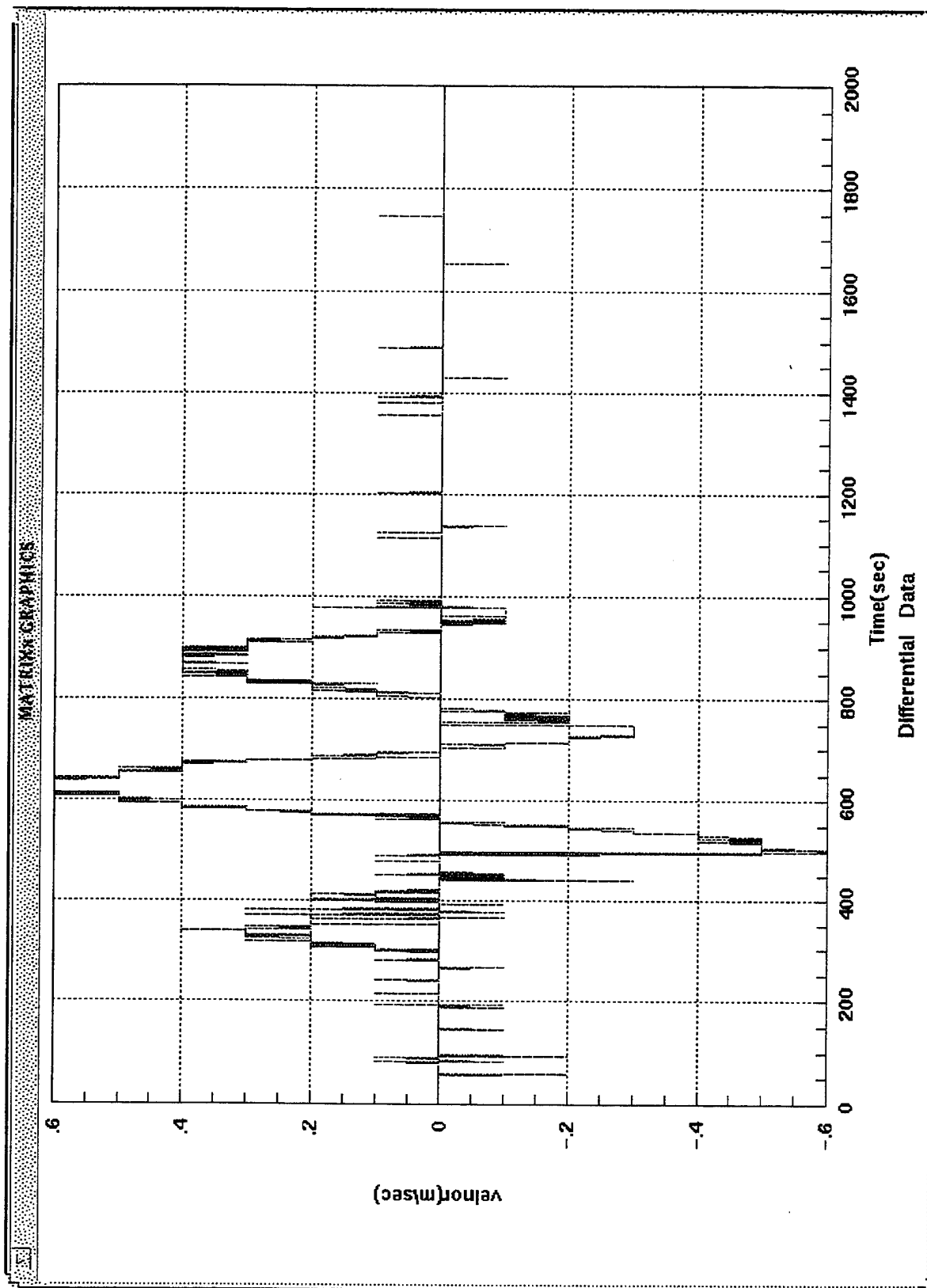


Fig.7.12 Tangent Plane coordinates, North velocity component, differential setup.

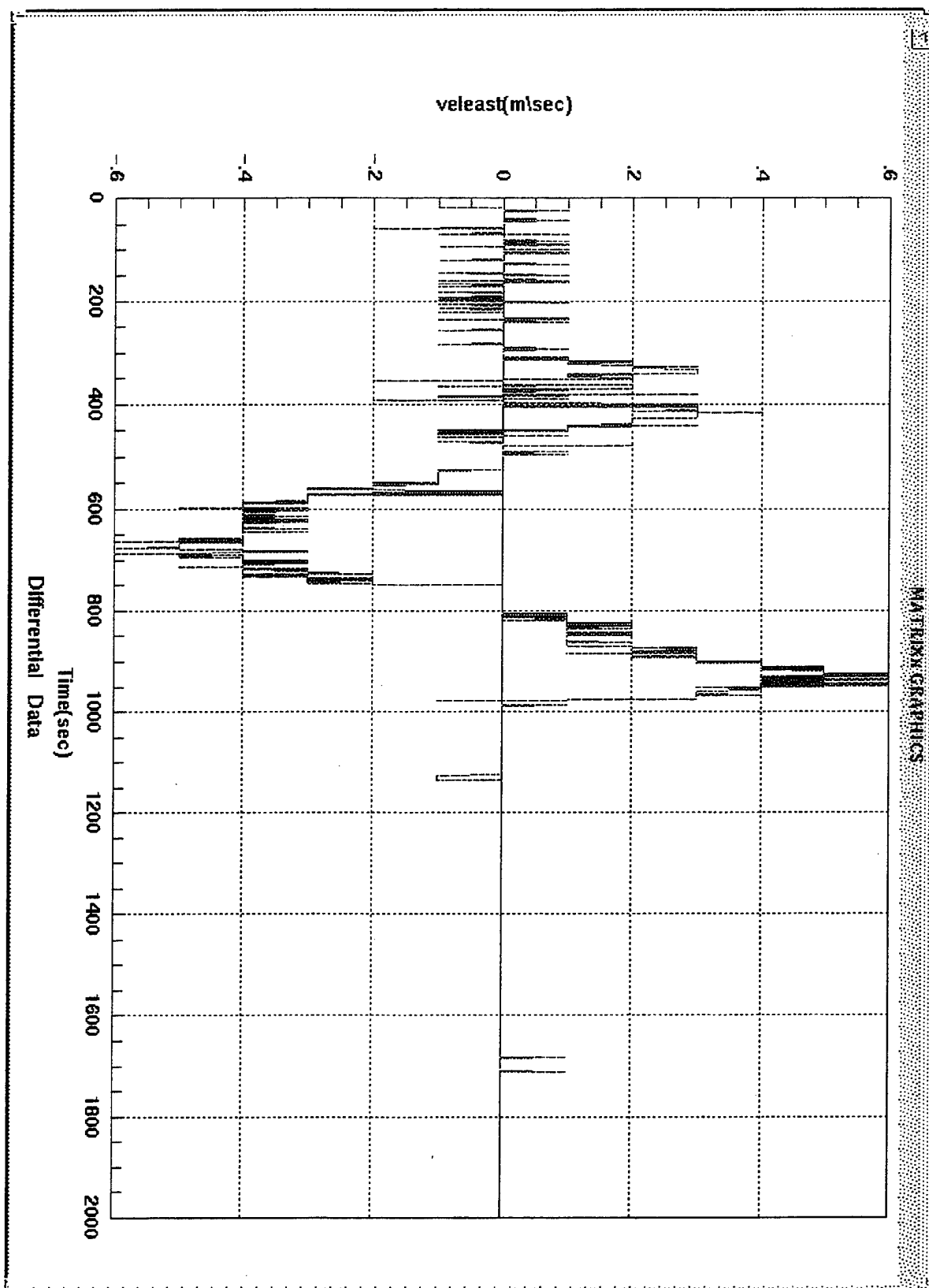


Fig.7.13 Tangent Plane coordinates, East velocity component, differential setup.

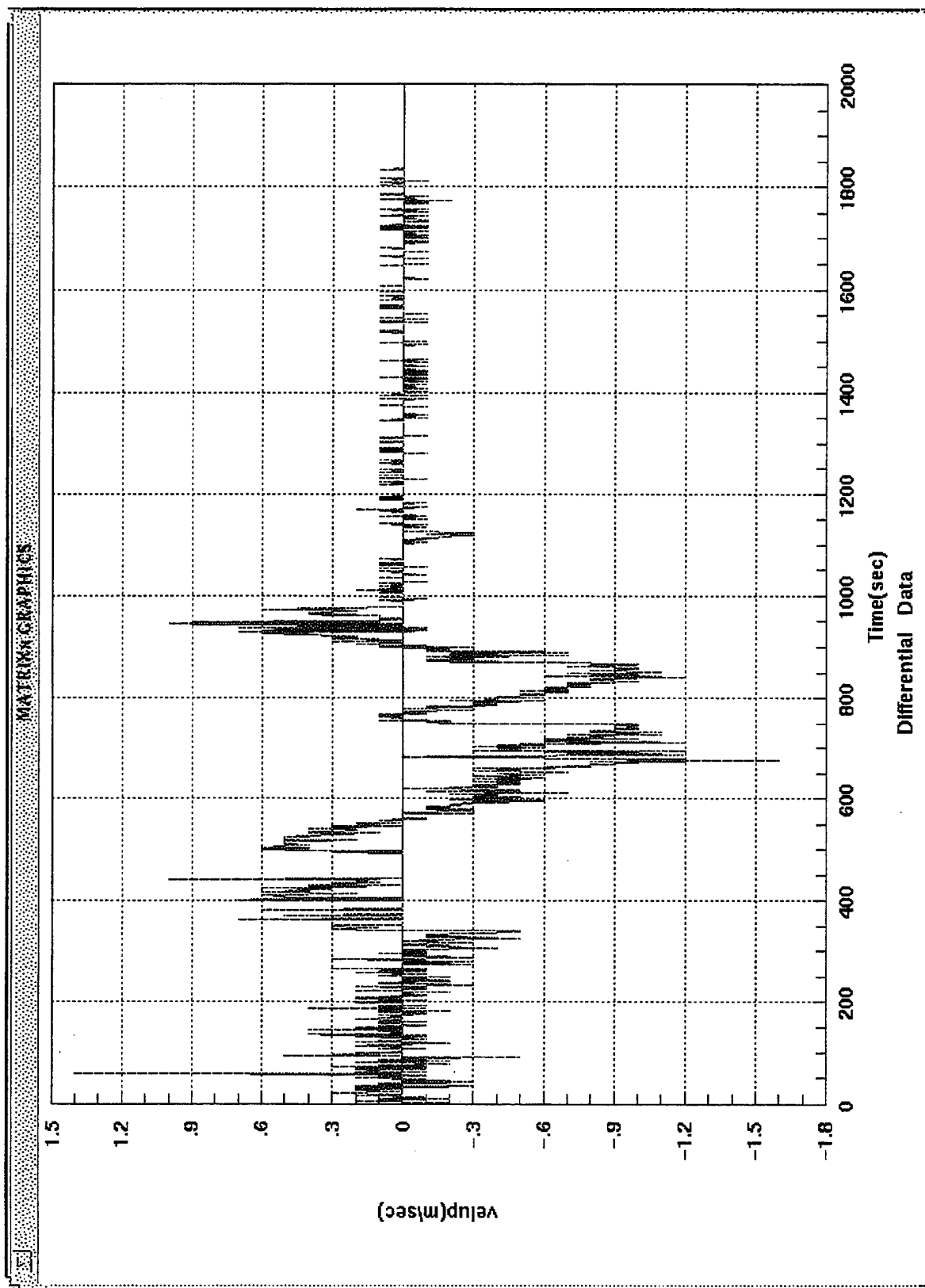


Fig. 7.14 Tangent Plane coordinates, Up velocity component, differential setup.

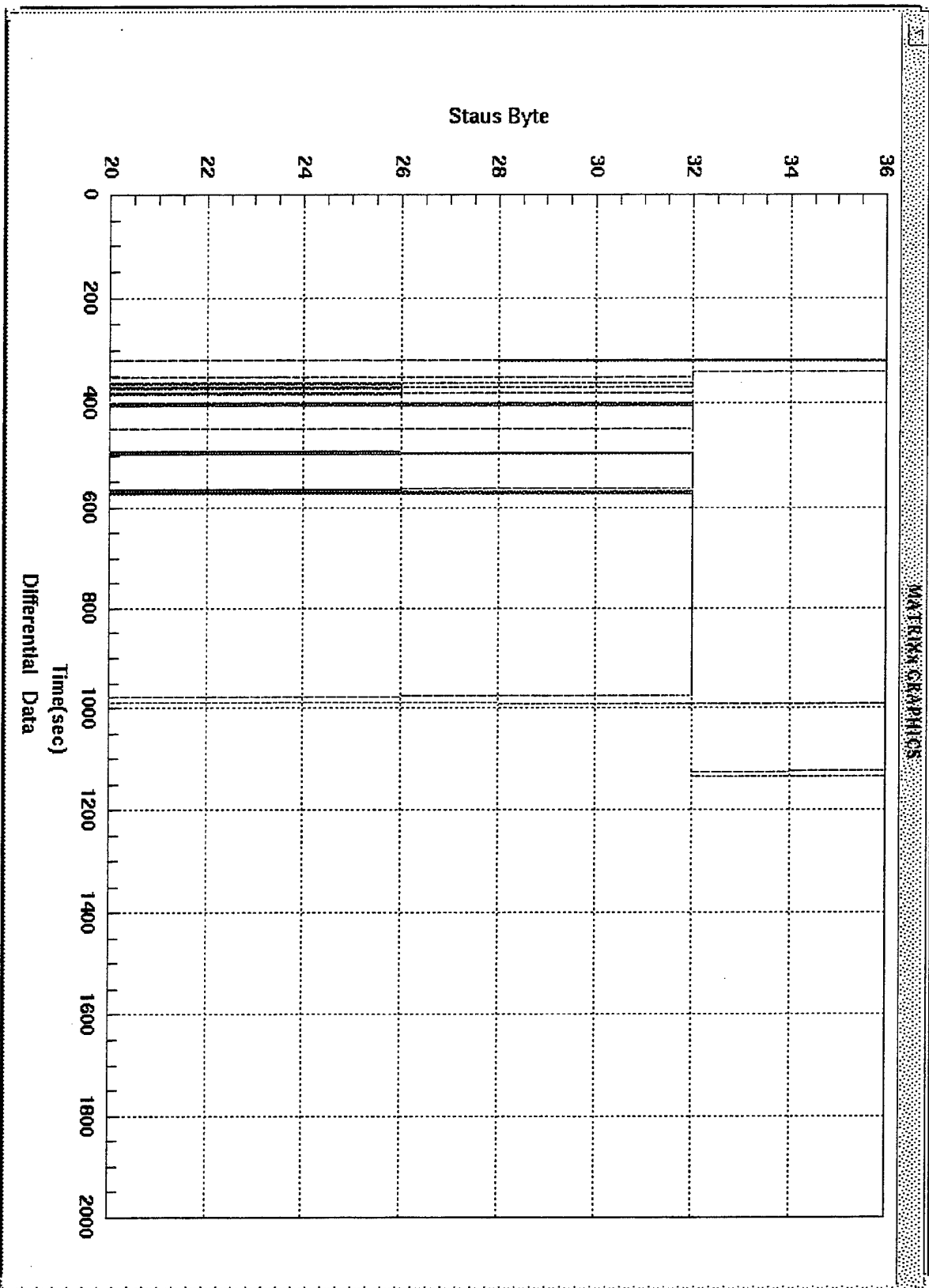


Fig. 7.15 The receiver's Status Byte, differential setup.

APPENDIX B. : TABLES

Table 4.1 PVT-6 specifications From Ref.2

Parameter	Description
Dynamics	<ul style="list-style-type: none"> • Velocity 1000 Knots (514.4 m/sec) • Acceleration 4 g • Jerk 5m/s³
Antenna	<ul style="list-style-type: none"> • Active microstrip patch Antenna Module • Powered by Receiver Module (5 Vdc @ 25 mA)
Antenna to Receiver Interconnection	<ul style="list-style-type: none"> • Single coaxial cable (6dB max loss at L1; 1575.42 MHz) • Typical length with RG58 coaxial cable; 20 feet (6 meters)
Serial Output	<ul style="list-style-type: none"> • RS232C Interface
Accuracy	<ul style="list-style-type: none"> • Less than 25 meters. SEP (without SA) • *DoD may invoke Selective Availability (SA), potentially degrading accuracy to 100 meters (2d RMS)
Altitude	<ul style="list-style-type: none"> • 60,000 feet (18 Km) (maximum)
Operating Temperature	<ul style="list-style-type: none"> • Active Antenna -40°C to +100°C (-40°F to +212°F) • Receiver Module -30°C to +80°C (-22°F to 176°F)
Humidity	<ul style="list-style-type: none"> • 95% non-condensing +30°C to +60°C (+86°F to +140°F)
Physical Dimensions	<ul style="list-style-type: none"> • Receiver PCB 3.94 X 2.76 X 0.65 in (100 X 70 X 16.5 mm) • Antenna Module 4.01 (dia) X 0.89 in (102 (dia) X 22.6 mm)
Weight	<ul style="list-style-type: none"> • Receiver and Housing 4.5 oz (128 g) • Antenna Module 4.8 oz (136.2 g)
Switched Power	<ul style="list-style-type: none"> • 9 to 16 Vdc or • 5 ± 0.25 Vdc
"Keep-Alive" BATT Power	<ul style="list-style-type: none"> • 4.75 – 16 Vdc; 0.3 mA
Power Consumption (typical)	<ul style="list-style-type: none"> • 1.3 W @ 5 Vdc input • 1.8 W @ 12 Vdc input
Receiver Interconnection	<ul style="list-style-type: none"> • AMP 10 pin, #104369-4 on Receiver Module <ul style="list-style-type: none"> - RS232C - Power input • OSX connector on Receiver and Antenna
MTBF (Mean Time Before Failure)	<ul style="list-style-type: none"> • 55,000 hours (estimated)

Table 5.1 PVT-6 basic output messages. From Ref.2

OUTPUT MESSAGE TYPE	CONTINUOUS (m=1...255)	ONE TIME (m=0)
Position/Channel Status	At selected update rate	When requested
Satellite Range Data Output*	At selected update rate	When requested
Pseudorange Correction Output*	At selected update rate	When requested
Ephemeris Data Output	When Eph data changes	When requested
Visible Satellite Status	When Vis data changes	When requested
DOP Table Status	When DOP data changes	When requested
Almanac Status	When Alm status changes	When requested
Almanac Data Output	When Alm data changes	When requested
Leap Second Pending		When Requested

*These output messages are available only as an option.

Table 5.2 PVT-6 differential capability. From Ref.2

FORMAT	TYPE	BAUD	BITS	START/ STOP	PARITY	FEATURES	DIFFERENTIAL CAPABILITY
Motorola	Binary	9600	8	1/1	No	Full Control/All Data	RTCM SC-104 ¹ and Custom ²
NMEA-0183	ASCII	4800	8	1/1	No	Partial Control/ Selected Mes- sages	RTCM SC-104 ¹
LORAN Emulation	ASCII	1200	8	1/1	No	Little Control/ 1 Output Message	None

Notes: 1. RTCM SC-104 decoding of Message Type #1 exists in de-optioned units.
It is available to all users at no additional cost.

2. The custom differential capability is available as Option-B only.

APPENDIX C. : USER_SER.C-GPS PROJECT

```
/**          USER_SER.C      PROJECT GPS          ***/

/*****
**@ File       : user_ser.c
**@ Project    : Ac100/c30
**@ Edit level : 3
**@
**@ Abstract:   : File contains functions which the user
**@              must define to interface with IP-SERIAL device
**@              driver.
**@              The functions must be called
**@
**@              get_SERIAL_parameters
**@              user_poll_SERIAL_out
**@              user_sample_SERIAL_in
**@
**@              Templates for the functions are provided.
**@
**@              get_SERIAL_parameters
**@              Function sets the asynchronous communication
**@              parameters for the IP-SERIAL module. Ring buffer
**@              sizes used to store received data must also be
**@              specified.
**@
**@              user_poll_SERIAL_out
**@              Function is called with the floating-point
**@              model output vector for the current sampling interval.
**@              The user is responsible for creating the transmit
**@              byte array and calling function write_serial which
**@              transmits the byte array across the serial channel.
**@
**@              user_sample_SERIAL_in
**@              Function is called every sampling interval. The
**@              user is responsible for filling the floating-point
**@              vector which is used as input to the model for
**@              the current sampling interval.
**@
**@ Modifications:
**@ -----
**@      Creation : 7-1--93   Henry Tominaga
**@      Revised  : 8-23-93   Brent Roman
**@      Revised  : 11-18-93  Steve Lynch
**@      Revised  : 2-27-95   Christofis/Hallberg
**@      Revised  : 4-27-95   Christofis
**/

#include <stddef.h>
#include <stdlib.h>
#include "sa_types.h"
#include "types.h"
#include "ioerrs.h"
#include "errcodes.h"
```

```

#include "iodriver.h"
#include "ipserial.h"
#include "printx.h"
#include <math.h>

#define NULL 0

/* semaphores and serial parameters for each physical channel */
private struct
{ boolean allSent; boolean broken; unsigned baud;} line_state[2];

struct user_type
{
    int update_interval;
    int update_count;
};

boolean          first_frame = true;
float            last_float[77], sol=299792458.0, llf0=1575420000.0, k2;
unsigned long    icpold1, icpold2, icpold3, icpold4, icpold5, icpold6;

/*****
*****
@@ THE INTERNALS OF THE FOLLOWING THREE FUNCTIONS MUST
@@ BE PROVIDED BY THE USER. PLEASE TAILOR FOR YOUR
@@ OWN SPECIFIC APPLICATION.
*****
*****/
/*****
Please define serial communication parameters and ring buffer
sizes in this function.
The communication parameters that must be set are the
standard asynchronous (RS-232) communication parameters.

The parameters are
    parity : NONE, EVEN, or ODD;
    baud_rate : enter any standard baud rate;
    stop_bits: ONE, TWO, or ONE_AND_HALF;
    transmit_data_size: 5, 6, 7, or 8 (bits);
    receive_data_size: 5, 6, 7, or 8 (bits);
    clock_multiplier: 1, 16, 32, or 64.
Please follow syntax given in the template function.
Please set parameters for the channels used.

Please also set the ring buffer size to be used for the
receive buffer. The size should exceed the maximum
anticipated number of bytes that will arrive at the
IP-Serial Module during one sampling interval.
*****/
public void get_SERIAL_parameters
(unsigned int          hardware_channel,
volatile struct user_param *device_param,
volatile struct ring_buffer_param *rec_buffer,
IOdevice              *device)
{
    struct user_type *user_ptr;
    serial_param_type *serptr;

    serptr = device->parameters;

    if (SERIAL_USER_PTR == NULL)
    {
        SERIAL_USER_PTR = (struct user_type *)malloc(sizeof(struct user_type));
        user_ptr          = SERIAL_USER_PTR;
    }

```

```

user_ptr->update_interval = 1000;
user_ptr->update_count    = 0;

```

```

}

```

```

if (hardware_channel == chanA || hardware_channel == chanB)

```

```

{

```

```

/* set parameters for channel A (IP-SERIAL channel 1)*/

```

```

device_param->parity      = NONE;

```

```

device_param->baud_rate    = 9600;

```

```

device_param->stop_bits    = ONE;

```

```

device_param->transmit_data_size = 8;

```

```

device_param->receive_data_size = 8;

```

```

device_param->clock_multiplier = 16;

```

```

/* set size for receive ring buffer */

```

```

rec_buffer->buffer_size    = 8000;

```

```

}

```

```

else

```

```

{

```

```

    printf("INVALID CHANNEL\n");

```

```

}

```

```

} /* get_SERIAL_parameters */

```

```

/*****

```

```

@@ Function : user_poll_SERIAL_out

```

```

@@ Abstract : Please specify user-defined serial output

```

```

@@           function here. Function called as

```

```

@@           scheduled output event.

```

```

@@

```

```

@@ Inputs   : sysptr      (user-transparent pointer to

```

```

@@             system attributes; DO NOT MANIPULATE)

```

```

@@             model_float[] (SystemBuild output vector)

```

```

@@             ser_channel (serial channel)

```

```

@@

```

```

@@ Outputs  :

```

```

@@

```

```

@@ Returns  : error status

```

```

@@

```

```

*****/

```

```

public RetCode user_SERIAL_out(IOdevice *device,
                                float model_float[],
                                u_int ser_channel)

```

```

{

```

```

    /*****

```

```

    * Given floating point model output, please create      *

```

```

    * buffer which contains bytes to be transmitted across  *

```

```

    * serial channel.                                         *

```

```

    *****/

```

```

    /*****

```

```

    * Fill the output buffer with data to be transmitted    *

```

```

    * by the background portion of the serial driver         *

```

```

    *****/

```

```

    return OK;

```

```

} /* user_SERIAL_out */

```

```

/*****

```

```

@@ Function : user_sample_SERIAL_in

```

```

@@ Abstract : Please specify user-defined serial input

```

```

@@           function here. Function called as

```

```

@@           scheduled input event.

```

```

@@

```

```

@@ Inputs   : sysptr      (user-transparent pointer to

```

```

@@             system attributes; DO NOT MANIPULATE)

```

```

@@          ser_channel (serial channel)
@@
@@ Outputs  : *model_float (pass back floating pt vector)
@@
@@ Returns  : error status
@@
*****/

```

```

public RetCode user_sample_SERIAL_in(IOdevice *device,
                                     float model_float[],
                                     u_int ser_channel)

{

    unsigned char    item[1], msg1[61], msg2[115], checksum1[1], checksum2[1],
                    mychecksum1, mychecksum2, status, model, mode2, mode3,
                    mode4, mode5, mode6, msg3[45], checksum3[1],
                    mychecksum3, msg4[62], checksum4[1], mychecksum4;

    struct user_type  *user_ptr;

    unsigned long     i, found = 0, icp1, icp2, icp3, icp4, icp5, icp6,
                    eph1, eph2, eph3, eph4, eph5, eph6;

    long              lat, lon, vel, heigps, heims1;

    int               num1, num2, num3, sat1, sat2, sat3, sat4, sat5, sat6,
                    satt1, satt2, satt3, satt4, satt5, satt6, num4,
                    velnor, velup, veleast;

    float             psrng1, psrng2, psrng3, psrng4, psrng5, psrng6,
                    psrngrate1, psrngrate2, psrngrate3, psrngrate4,
                    psrngrate5, psrngrate6, loctime, sattime1, sattime2,
                    sattime3, sattime4, sattime5, sattime6, ecefx, ecefy,
                    ecefz, velnor1, veleast1, velup1, hd, timeref,
                    pscor1, pscor2, pscor3, pscor4, pscor5, pscor6,
                    psrkor1, psrkor2, psrkor3, psrkor4, psrkor5, psrkor6,
                    locsec, locsecf,
                    satsec1, satsec2, satsec3, satsec4,
                    satsec5, satsec6, satsecf1, satsecf2, satsecf3,
                    satsecf4, satsecf5, satsecf6, heigps1, heims1,
                    vel1, lat1, lon1, ecefx1, ecefy1, ecefz1;

    /*****
    * set user pointer to buffer allocated by get parameters *
    * this buffer is passed around with the structure device *
    * and should only be accessed via the SERIAL_USER_PTR *
    * define *
    *****/

    user_ptr          = SERIAL_USER_PTR;

    if(first_frame)
    {

```

```

for(i=0;i<77;i++)
last_float[i]=7.0;
first_frame=false;
icpold1=0.0; icpold2=0.0; icpold3=0.0; icpold4=0.0;
icpold5=0.0; icpold6=0.0;
}

```

```

/***** READING CHANNEL A *****/

```

```

/*-----@@Ba-----*/

```

```

if (ser_channel==1)
{

```

```

    num1 = numbytes_in_buffer(device->parameters);

```

```

    if (num1 > 67)
    {

```

```

        while(found!=4)
        {

```

```

            read_serial(device->parameters,1,item);
            switch(item[0])
            {

```

```

                case '@':if(found==0)
                    found=1;
                else if (found==1)
                    found=2;
                else
                    found=0;
                break;
                case 'B':if (found==2)
                    found=3;
                else
                    found=0;
                break;
                case 'a':if (found==3)
                    found=4;
                else
                    found=0;
                break;

```

```

            default:found=0;
            }/*end switch*/
        }/*end while*/

```

```

/*At this point we have read a '@@Ba' message*/

```

```

read_serial (device->parameters, 61, msg1);
read_serial (device->parameters,1, checksum1);

```

```

/*Convert message bytes to numbers*/

```

```

lat = msg1[11]*0x1000000 + msg1[12]*0x10000 + msg1[13]*0x100 + msg1[14];

```

```

lon = msg1[15]*0x1000000 + msg1[16]*0x10000 + msg1[17]*0x100 + msg1[18];

```

```

heigps =msg1[19]*0x1000000 +msg1[20]*0x10000 +msg1[21]*0x100 + msg1[22];

```

```

heims1 = msg1[23]*0x1000000 +msg1[24]*0x10000 +msg1[25]*0x100 +msg1[26];
vel = msg1[27]*0x100 + msg1[28];
hd = msg1[29]*0x100 + msg1[30];
hd = hd/10.0;
status = msg1[60];
vell=vel/100.0; heigps1=heigps/100.0; heims11=heims1/100.0;
lat1=lat/100.0; lon1=lon/100.0;

```

```

/*check if message is correct*/

```

```

mychecksum1='B'^'a';
for(i=0;i<61;i++)
mychecksum1^=msg1[i];

```

```

    if(mychecksum1!=checksum1[0])
    for (i = 0; i < 9; i++)
    {
        model_float[i] = last_float[i];
    }
    else
    {
        model_float[0]=lat1;
        model_float[1]=lon1;
        model_float[2]=heigps1;
        model_float[3]=heims11;
        model_float[4]=vell;
        model_float[5]=hd;
        model_float[6]=status;
        model_float[7]=checksum1[0];
        model_float[8]=mychecksum1;

        for (i = 0; i < 9; i++)
        {
            last_float[i] = model_float[i];
        }

    }/*endelse*/
} /*endif*/

```

```

if(num1<=67)
{
    for (i = 0; i < 9; i++)
    {
        model_float[i] = last_float[i];
    }
}

```

```

/*-----@Bk-----*/

```

```
found=0;
num4 = numbytes_in_buffer(device->parameters);
```

```
if (num4 > 68)
{
    while(found!=4)
    {
        read_serial(device->parameters,1,item);
        switch(item[0])
        {
            case '@':if(found==0)
                        found=1;
                    else if (found==1)
                        found=2;
                    else
                        found=0;
                    break;
            case 'B':if (found==2)
                        found=3;
                    else
                        found=0;
                    break;
            case 'k':if (found==3)
                        found=4;
                    else
                        found=0;
                    break;

            default:found=0;
        }/*end switch*/
    }/*end while*/
}
```

```
/*At this point we have read a '@@Bk' message*/
```

```
read_serial (device->parameters, 62, msg4);
read_serial (device->parameters, 1, checksum4);
```

```
/*Convert message bytes to numbers*/
```

```
ecefx  =msg4[20]*0x1000000+msg4[21]*0x10000+msg4[22]*100+msg4[23];
ecify  =msg4[24]*0x1000000+msg4[25]*0x10000+msg4[26]*100+msg4[27];
ecfz   =msg4[28]*0x1000000+msg4[29]*0x10000+msg4[30]*100+msg4[31];
ecfx1  =ecfx/10.0;
ecfy1  =ecfy/10.0;
ecfz1  =ecfz/10.0;
velnor =msg4[12]*0x100+msg4[13];
veleat =msg4[14]*0x100+msg4[15];
velup  =msg4[16]*0x100+msg4[17];
```

```

if(velnor>5000)
velnor=velnor-65536;

if(veleast>5000)
veleast=veleast-65536;

if(velup>5000)
velup=velup-65536;

velnor1  =velnor/10.0;

veleast1 =veleast/10.0;

velup1   =velup/10.0;


/*check if message is correct*/


mychecksum4='B'^'k';
for(i=0;i<62;i++)
mychecksum4^=msg4[i];

if(mychecksum4!=checksum4[0])
{
    for (i = 69; i < 77; i++)
    {
        model_float[i-27] = last_float[i];
    }
}
else
{

    model_float[42] =ecef_x1;
    model_float[43] =ecef_y1;
    model_float[44] =ecef_z1;
    model_float[45] =velnor1;
    model_float[46] =veleast1;
    model_float[47] =velup1;
    model_float[48] =checksum4[0];
    model_float[49] =mychecksum4;


    for (i = 42; i < 50; i++)
    {
        last_float[i+27] = model_float[i];
    }
}

```



```

        }/*endelse*/

    }/*endif*/


if(num4<=68)
{
    for (i = 69; i < 77; i++)
    {
        model_float[i-27] = last_float[i];
    }

}

/*-----@@Bg-----*/


found=0;
num2 = numbytes_in_buffer(device->parameters);


if (num2 > 121)
{
    while(found!=4)
    {
        read_serial(device->parameters,1,item);
        switch(item[0])
        {
            case '@':if(found==0)
                        found=1;
                    else if (found==1)
                        found=2;
                    else
                        found=0;
                    break;
            case 'B':if (found==2)
                        found=3;
                    else
                        found=0;
                    break;
            case 'g':if (found==3)
                        found=4;
                    else
                        found=0;
                    break;

            default:found=0;
        }/*end switch*/
    }/*end while*/


/*At this point we have read a '@@Bg' message*/

read_serial (device->parameters, 115, msg2);
read_serial (device->parameters,1, checksum2);

/*Convert message bytes to numbers*/

```

```
locsec =msg2[0]*0x10000 +msg2[1]*0x100 +msg2[2];
```

```
locsecf =msg2[3]*0x1000000 +msg2[4]*0x10000 +msg2[5]*0x100 +msg2[6];
```

```
/* GPS channel1 data */
```

```
sat1 =msg2[7];
```

```
model =msg2[8];
```

```
satsec1 =msg2[9]*0x10000 +msg2[10]*0x100 +msg2[11];
```

```
satsecf1=msg2[12]*0x1000000 +msg2[13]*0x10000 +msg2[14]*0x100 +msg2[15];
```

```
icp1 =msg2[16]*0x1000000 +msg2[17]*0x10000 +msg2[18]*0x100 +msg2[19];
```

```
/* GPS channel2 data */
```

```
sat2 =msg2[25];
```

```
mode2 =msg2[26];
```

```
satsec2 =msg2[27]*0x10000 +msg2[28]*0x100 +msg2[29];
```

```
satsecf2=msg2[30]*0x1000000 +msg2[31]*0x10000 +msg2[32]*0x100 +msg2[33];
```

```
icp2 =msg2[34]*0x1000000 +msg2[35]*0x10000 +msg2[36]*0x100 +msg2[37];
```

```
/* GPS channel3 data */
```

```
sat3 =msg2[43];
```

```
mode3 =msg2[44];
```

```
satsec3 =msg2[45]*0x10000 +msg2[46]*0x100 +msg2[47];
```

```
satsecf3=msg2[48]*0x1000000 +msg2[49]*0x10000 +msg2[50]*0x100 +msg2[51];
```

```
icp3 =msg2[52]*0x1000000 +msg2[53]*0x10000 +msg2[54]*0x100 +msg2[55];
```

```
/* GPS channel4 data */
```

```
sat4 =msg2[61];
```

```
mode4 =msg2[62];
```

```
satsec4 =msg2[63]*0x10000 +msg2[64]*0x100 +msg2[65];
```

```
satsecf4=msg2[66]*0x1000000 +msg2[67]*0x10000 +msg2[68]*0x100 +msg2[69];
```

```
icp4 =msg2[70]*0x1000000 +msg2[71]*0x10000 +msg2[72]*0x100 +msg2[73];
```

```
/* GPS channel5 data */
```

```
sat5 =msg2[79];
```

```
mode5 =msg2[80];
```

```
satsec5 =msg2[81]*0x10000 +msg2[82]*0x100 +msg2[83];
```

```
satsecf5=msg2[84]*0x1000000 +msg2[85]*0x10000 +msg2[86]*0x100 +msg2[87];
```

```
icp5 =msg2[88]*0x1000000 +msg2[89]*0x10000 +msg2[90]*0x100 +msg2[91];
```

```
/* GPS channel6 data */
```

```
sat6 =msg2[97];
```

```
mode6 =msg2[98];
```

```
satsec6 =msg2[99]*0x10000 +msg2[100]*0x100 +msg2[101];
```

```
satsecf6=msg2[102]*0x1000000+msg2[103]*0x10000+msg2[104]*0x100+msg2[105];
```

```
icp6=msg2[106]*0x1000000 +msg2[107]*0x10000 +msg2[108]*0x100 +msg2[109];
```

```
/*check if message is correct*/
```

```
mychecksum2='B'^'g';
```

```
for(i=0;i<115;i++)
```

```
mychecksum2^=msg2[i];
```

```
if(mychecksum2!=checksum2[0])
```

```
{
```

```
    for (i = 9; i < 42; i++)
```

```
    {
```

```
        model_float[i] = last_float[i];
```

```
    }
```

```
}
```

```
else
```

```
{
```

```
    /*Satellite and receiver GPS time (sec)*/
```

```
    loctime =locsec +locsecf/(pow(10.0,9.0));
```

```
    sattime1 =satsec1 +satsecf1/(pow(10.0,9.0));
```

```
    sattime2 =satsec2 +satsecf2/(pow(10.0,9.0));
```

```
    sattime3 =satsec3 +satsecf3/(pow(10.0,9.0));
```

```
    sattime4 =satsec4 +satsecf4/(pow(10.0,9.0));
```

```
    sattime5 =satsec5 +satsecf5/(pow(10.0,9.0));
```

```
    sattime6 =satsec6 +satsecf6/(pow(10.0,9.0));
```

```
    /*Pseudoranges (m)*/
```

```
    psrng1 =(loctime - sattime1)*sol;
```

```
    psrng2 =(loctime - sattime2)*sol;
```

```
    psrng3 =(loctime - sattime3)*sol;
```

```
    psrng4 =(loctime - sattime4)*sol;
```

```
    psrng5 =(loctime - sattime5)*sol;
```

```
    psrng6 =(loctime - sattime6)*sol;
```

```
    k2=sol/(11f0*65536.0);
```

```
    /*Pseudorange rates (m/sec)*/
```

```
    psrngrate1 =k2*(icp1 - icpold1);
```

```
    psrngrate2 =k2*(icp2 - icpold2);
```

```
    psrngrate3 =k2*(icp3 - icpold3);
```

```
    psrngrate4 =k2*(icp4 - icpold4);
```

```
psrngrate5 =k2*(icp5 - icpold5);
psrngrate6 =k2*(icp6 - icpold6);
```

```
icpold1 =icp1;
icpold2 =icp2;
icpold3 =icp3;
icpold4 =icp4;
icpold5 =icp5;
icpold6 =icp6;
```

```
model_float[9]  = loctime;
model_float[10] = sat1;
model_float[11] = model;
model_float[12] = sattime1;
model_float[13] = psrng1;
model_float[14] = psrngrate1;
model_float[15] = sat2;
model_float[16] = mode2;
model_float[17] = sattime2;
model_float[18] = psrng2;
model_float[19] = psrngrate2;
model_float[20] = sat3;
model_float[21] = mode3;
model_float[22] = sattime3;
model_float[23] = psrng3;
model_float[24] = psrngrate3;
model_float[25] = sat4;
model_float[26] = mode4;
model_float[27] = sattime4;
model_float[28] = psrng4;
model_float[29] = psrngrate4;
model_float[30] = sat5;
model_float[31] = mode5;
model_float[32] = sattime5;
model_float[33] = psrng5;
model_float[34] = psrngrate5;
model_float[35] = sat6;
model_float[36] = mode6;
model_float[37] = sattime6;
model_float[38] = psrng6;
model_float[39] = psrngrate6;
model_float[40] = checksum2[0];
model_float[41] = mychecksum2;
```

```
for (i = 9; i < 42; i++)
{
    last_float[i] = model_float[i];
}
```

```
/*endif*/
```

```
/*endif*/
```

```
if(num2<=121)
{
```

```

    for (i = 9; i < 42; i++)
    {
        model_float[i] = last_float[i];
    }
}

```

```

}/*endif ser_channel1*/

```

```

./***** READING CHANNEL B *****/

```

```

/*-----@@Ce-----*/

```

```

if (ser_channel==0)
{

```

```

    found=0;
    num3 = numbytes_in_buffer(device->parameters);

```

```

    if (num3 > 51)

```

```

    {
        while(found!=4)
        {
            read_serial(device->parameters,1,item);
            switch(item[0])
            {
                case '@':if(found==0)
                            found=1;
                        else if (found==1)
                            found=2;
                        else
                            found=0;
                        break;
                case 'C':if (found==2)
                            found=3;
                        else
                            found=0;
                        break;
                case 'e':if (found==3)
                            found=4;
                        else
                            found=0;
                        break;

                default:found=0;
            }/*end switch*/
        }/*end while*/
    }

```

```
/*At this point we have read a '@@Ce' message*/
```

```
read_serial (device->parameters, 45, msg3);  
read_serial (device->parameters,1, checksum3);
```

```
/*Convert message bytes to numbers*/
```

```
/*GPS time reference*/
```

```
timeref =msg3[0]*0x10000 +msg3[1]*0x100 +msg3[2];
```

```
/* Differential corections */
```

```
/* Channel1 data */
```

```
satt1 =msg3[3];  
pscor1 =msg3[4]*0x10000 +msg3[5]*0x100 +msg3[6];  
psrcor1 =msg3[7]*0x100 +msg3[8];  
eph1 =msg3[9];
```

```
/* Channel2 data */
```

```
satt2 =msg3[10];  
pscor2 =msg3[11]*0x10000 +msg3[12]*0x100 +msg3[13];  
psrcor2 =msg3[14]*0x100 +msg3[15];  
eph2 =msg3[16];
```

```
/* Channel3 data */
```

```
satt3 =msg3[17];  
pscor3 =msg3[18]*0x10000 +msg3[19]*0x100 +msg3[20];  
psrcor3 =msg3[21]*0x100 +msg3[22];  
eph3 =msg3[23];
```

```
/* Channel4 data */
```

```
satt4 =msg3[24];  
pscor4 =msg3[25]*0x10000 +msg3[26]*0x100 +msg3[27];  
psrcor4 =msg3[28]*0x100 +msg3[29];  
eph4 =msg3[30];
```

```
/* Channel5 data */
```

```
satt5 =msg3[31];  
pscor5 =msg3[32]*0x10000 +msg3[33]*0x100 +msg3[34];  
psrcor5 =msg3[35]*0x100 +msg3[36];  
eph5 =msg3[37];
```

```
/* Channel6 data */
```

```
satt6 =msg3[38];  
pscor6 =msg3[39]*0x10000 +msg3[40]*0x100 +msg3[41];  
psrcor6 =msg3[42]*0x100 +msg3[43];  
eph6 =msg3[44];
```

```
/*check if message is correct*/
```

```
mychecksum3='C'^'e';  
for(i=0;i<45;i++)  
mychecksum3^=msg3[i];
```

```
if(mychecksum3!=checksum3[0])
```

```
{  
  
    for (i = 42; i < 69; i++)  
    {
```

```

        model_float[i-42] = last_float[i];
    }

else
{
    timeref =timeref/10.0;

    /*Pseudorange correction (m)*/

    pscor1 =pscor1/(pow(10,2));
    pscor2 =pscor2/(pow(10,2));
    pscor3 =pscor3/(pow(10,2));
    pscor4 =pscor4/(pow(10,2));
    pscor5 =pscor5/(pow(10,2));
    pscor6 =pscor6/(pow(10,2));

    /*Pseudorange rate correction (m/sec)*/

    psrkor1 =psrkor1/(pow(10,3));
    psrkor2 =psrkor2/(pow(10,3));
    psrkor3 =psrkor3/(pow(10,3));
    psrkor3 =psrkor3/(pow(10,3));
    psrkor3 =psrkor3/(pow(10,3));
    psrkor4 =psrkor4/(pow(10,3));

    model_float[0] =timeref;
    model_float[1] =satt1;
    model_float[2] =pscor1;
    model_float[3] =psrkor1;
    model_float[4] =eph1;
    model_float[5] =satt2;
    model_float[6] =pscor2;
    model_float[7] =psrkor2;
    model_float[8] =eph2;
    model_float[9] =satt3;
    model_float[10] =pscor3;
    model_float[11] =psrkor3;
    model_float[12] =eph3;
    model_float[13] =satt4;
    model_float[14] =pscor4;
    model_float[15] =psrkor4;
    model_float[16] =eph4;
    model_float[17] =satt5;
    model_float[18] =pscor5;
    model_float[19] =psrkor5;
    model_float[20] =eph5;
    model_float[21] =satt6;
    model_float[22] =pscor6;
    model_float[23] =psrkor6;
    model_float[24] =eph6;
    model_float[25] =checksum3[0];
    model_float[26] =mychecksum3;

    for (i = 42; i < 69; i++)
    {
        last_float[i] = model_float[i-42];
    }

}/*endelse*/

```

```
 */*endif*/
```

```
if(num3<=51)
```

```
{  
    for (i = 42; i < 69; i++)  
    {  
        model_float[i-42] = last_float[i];  
    }  
}
```

```
 */*endif ser_channel0*/
```

```
return OK;
```

```
 } /* user_sample_SERIAL_in */
```

```
 /*****  
      END OF FILE  
 *****/
```


APPENDIX D. : USER_SER.C-BK PROJECT

```

/****          USER_SER.C          PROJECT bk          ****/

/*****
**@ File       : user_ser.c
**@ Project    : Ac100/c30
**@ Edit level : 3
**@
**@ Abstract:   : File contains functions which the user
**@              must define to interface with IP-SERIAL device
**@              driver.
**@              The functions must be called
**@
**@              get_SERIAL_parameters
**@              user_poll_SERIAL_out
**@              user_sample_SERIAL_in
**@
**@              Templates for the functions are provided.
**@
**@              get_SERIAL_parameters
**@                  Function sets the asynchronous communication
**@                  parameters for the IP-SERIAL module. Ring buffer
**@                  sizes used to store received data must also be
**@                  specified.
**@
**@              user_poll_SERIAL_out
**@                  Function is called with the floating-point
**@                  model output vector for the current sampling interval.
**@                  The user is responsible for creating the transmit
**@                  byte array and calling function write_serial which
**@                  transmits the byte array across the serial channel.
**@
**@              user_sample_SERIAL_in
**@                  Function is called every sampling interval. The
**@                  user is responsible for filling the floating-point
**@                  vector which is used as input to the model for
**@                  the current sampling interval.
**@
**@ Modifications:
**@ -----
**@      Creation : 7-1--93  Henry Tominaga
**@      Revised  : 8-23-93  Brent Roman
**@      Revised  : 11-18-93 Steve Lynch
**@      Revised  : 4-27-95  Christofis
**@
*/

#include <stddef.h>
#include <stdlib.h>
#include "sa_types.h"
#include "types.h"
#include "ioerrs.h"
#include "errcodes.h"
#include "iodriver.h"
#include "ipserial.h"
#include "printx.h"
```

```
#define NULL 0
```

```
/* semaphores and serial parameters for each physical channel */  
private struct  
{ boolean allSent; boolean broken; unsigned baud;} line_state[2];
```

```
struct user_type  
{  
    int update_interval;  
    int update_count;
```

```
};
```

```
/*  
*****  
*****  
@@ THE INTERNALS OF THE FOLLOWING THREE FUNCTIONS MUST  
@@ BE PROVIDED BY THE USER. PLEASE TAILOR FOR YOUR  
@@ OWN SPECIFIC APPLICATION.  
*****  
******/
```

```
/*  
*****  
Please define serial communication parameters and ring buffer  
sizes in this function.  
The communication parameters that must be set are the  
standard asynchronous (RS-232) communication parameters.
```

The parameters are

```
parity : NONE, EVEN, or ODD;  
baud_rate : enter any standard baud rate;  
stop_bits: ONE, TWO, or ONE_AND_HALF;  
transmit_data_size: 5, 6, 7, or 8 (bits);  
receive_data_size: 5, 6, 7, or 8 (bits);  
clock_multiplier: 1, 16, 32, or 64.
```

Please follow syntax given in the template function.
Please set parameters for the channels used.

Please also set the ring buffer size to be used for the
receive buffer. The size should exceed the maximum
anticipated number of bytes that will arrive at the
IP-Serial Module during one sampling interval.

```
*****/
```

```
public void get_SERIAL_parameters
```

```
(unsigned int hardware_channel,  
 volatile struct user_param *device_param,  
 volatile struct ring_buffer_param *rec_buffer,  
 IOdevice *device)
```

```
{  
    struct user_type *user_ptr;  
    serial_param_type *serptr;
```

```
    serptr = device->parameters;
```

```
    if (SERIAL_USER_PTR == NULL)
```

```
    {  
        SERIAL_USER_PTR = (struct user_type *)malloc(sizeof(struct user_type));  
        user_ptr = SERIAL_USER_PTR;  
        user_ptr->update_interval = 1000;  
        user_ptr->update_count = 0;  
    }
```

```
    if (hardware_channel == chanA || hardware_channel == chanB)
```

```
    {  
        /* set parameters for channel A (IP-SERIAL channel 1)*/
```

```

device_param->parity          = NONE;
device_param->baud_rate        = 9600;
device_param->stop_bits        = ONE;
device_param->transmit_data_size = 8;
device_param->receive_data_size = 8;
device_param->clock_multiplier = 16;
/* set size for receive ring buffer */
rec_buffer->buffer_size        = 1500;

```

```

}
else

```

```

{
    printf("INVALID CHANNEL\n");
}

```

```

} /* get_SERIAL_parameters */

```

```

/*****

```

```

@@ Function : user_poll_SERIAL_out

```

```

@@ Abstract : Please specify user-defined serial output
@@             function here.  Function called as
@@             scheduled output event.
@@

```

```

@@ Inputs    : sysptr          (user-transparent pointer to
@@                        system attributes; DO NOT MANIPULATE)
@@            model_float[] (SystemBuild output vector)
@@            ser_channel (serial channel)
@@

```

```

@@ Outputs   :
@@

```

```

@@ Returns   : error status
@@

```

```

*****/

```

```

    unsigned char    my_float[8], checksum=0x00;

```

```

public RetCode user_SERIAL_out(IOdevice *device,
                                float model_float[],
                                u_int ser_channel)
{

```

```

    checksum='B'^'k';
    checksum^=0x01;

```

```

    my_float[0]='@';
    my_float[1]='@';
    my_float[2]='B';
    my_float[3]='k';
    my_float[4]=0x01;
    my_float[5]=checksum;
    my_float[6]='\r';
    my_float[7]='\n';

```

```

/*****
*   Given floating point model output, please create
*   buffer which contains bytes to be transmitted across
*   serial channel.
*****/

```

```

/*****
* Fill the output buffer with data to be transmitted      *
* by the background portion of the serial driver          *
*****/

    write_serial(device->parameters,my_float,8);

    return OK;

} /* user_SERIAL_out */

/*****
@@ Function : user_sample_SERIAL_in
@@ Abstract : Please specify user-defined serial input
@@            function here.  Function called as
@@            scheduled input event.
@@
@@ Inputs    : sysptr      (user-transparent pointer to
@@                        system attributes; DO NOT MANIPULATE)
@@            ser_channel (serial channel)
@@
@@ Outputs   : *model_float (pass back floating pt vector)
@@
@@ Returns   : error status
@@
*****/
public RetCode user_sample_SERIAL_in(IOdevice *device,
                                     float model_float[],
                                     u_int ser_channel)

{
    struct user_type      *user_ptr;

    /*****
    * set user pointer to buffer allocated by get parameters *
    * this buffer is passed around with the structure device *
    * and should only be accessed via the SERIAL_USER_PTR    *
    * define                                                  *
    *****/

    user_ptr      = SERIAL_USER_PTR;

    return OK;

} /* user_sample_SERIAL_in */

```

END OF FILE
***** /

LIST OF REFERENCES

1. Logsdon, T., *The Navstar Global Positioning System*, Van Nostrand Reinhold, 1992.
2. Motorola, *Motorola GPS Technical Reference Manual*, Revision 4.3, July 1993.
3. Rockwell International Space Systems Division, *GPS Interface Control Document ICD-GPS-200*, (Revision B), 1987.
4. Texas Instruments, *TMS320C3x-User's Guide*, Revision J, October 1994.
5. Integrated Systems, *AC100 Model C30 Supplemental Reference* Version AC3.4, February 1994.
6. Twite, E., *Selection and Integration of a Global Positioning System Receiver with a CPU*, Thesis, Naval Postgraduate School, June 1994.
7. Hurn, J., *GPS a Guide to the Next Utility*, Trimble Navigation, 1987.
8. Dana, P., *Geodetic Datums Overview*, World Wide Web Home Page, University of Texas, February 1995.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
Cameron Station
Alexandria, Virginia 22304-6145
2. Library, Code 52 2
Naval Postgraduate School
Monterey, California 93943-5101
3. Doctor Isaac I. Kaminer, Code AA/KA 3
Department of Aeronautics and Astronautics
Naval Postgraduate School
Monterey, California 93943-5121
4. Colonel Athan Constantine 1
433 Lewis Road
Presidio of Monterey, California 93944
5. Lieutenant Junior Grade Alexandros Christofis 2
Cheronias 17
121-33 , Peristeri
ATHENS, GREECE